INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Registo de dados de exposição a risco

MIGUEL DA ROCHA RIBEIRO
Julho de 2019

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Registo de dados de exposição a risco

Miguel da Rocha Ribeiro

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Engenharia de Software

Orientador: António Manuel Cardoso Da Costa

Supervisor: Fernando Cavaleiro

Júri: Nuno Miguel Gomes Bettencourt, Professor Adjunto, ISEP

Presidente: Rosa Maria Do Nascimento Da Silva Reis, Professor Adjunto, ISEP

Porto, Julho 2019

Dedicatória

À minha família, amigos e namorada por todo o apoio e compreensão.

Resumo

Apesar da consciencialização para o tema de Segurança e Saúde no Trabalho e da evolução de medidas de prevenção de acidentes, o número de ocorrências de acidentes de trabalho continua a apresentar valores elevados [1]. Associado a este problema surge a necessidade de confirmar a ocorrência do sinistro às empresas prestadores de serviços de seguros, às quais se torna imperativo fornecer dados fiáveis e que confirmem a ocorrência do mesmo.

O produto da Ábaco Consultores, *inCloud for Safemed*, é uma solução destinada à área de Segurança e Saúde no Trabalho que necessitou de integrar novas funcionalidades para dar resposta ao problema apresentado. Assim, de forma a facilitar todo o processo pós acidente, os objetivos basearam-se no registo de dados que possam ter contribuído para a ocorrência do sinistro, permitindo assim confirmar a veracidade do mesmo.

Desta forma, a abordagem proposta foi o desenvolvimento de um sistema abstrato que permitiria registar de forma imutável e segura dados críticos associados a um possível sinistro, dados esses que serão utilizados para verificar as condições do acidente e evitar assim qualquer ação fraudulenta.

Assim, o presente documento apresenta a solução desenvolvida para o problema descrito, incluindo a pesquisa e avaliação de tecnologias, o *design* arquitetural do sistema e o desenvolvimento e testes de avaliação da mesma.

O desenvolvimento da solução permitiu solucionar o problema apresentado, garantindo o armazenamento seguro e imutável dos dados num sistema essencialmente baseado na tecnologia *Apache Kafka*. Contudo, como trabalho futuro é importante a otimização de determinados componentes, bem como a implementação de medidas de segurança adicionais e testes de avaliação num cenário real.

Palavras-chave: Saúde e Segurança no Trabalho, Acidentes de trabalho, *Blockchain*, Sistemas distribuídos, Armazenamento imutável, *Apache Kafka*, Encriptação

Abstract

Despite the awareness of Safety and Health at Work and the evolution of accident prevention measures, the number of accidents at work remains high [1]. Allied to this problem there is a need to confirm the occurrence of the accident to the companies that provide insurance services, to which it is imperative to provide reliable data that verify the sinister's occurrence.

The product of Ábaco Consultores, inCloud for Safemed, is a solution for Safety and Health at Work that needed to integrate new functionalities to respond to the presented problem. Thus, in order to simplify the entire post-accident process, the objectives were based on the recording of data that may have contributed to the occurrence of the accident, allowing confirmation of the accident's veracity.

Therefore, the proposed approach was the development of an abstract system that allowed to record in a immutable and secure way critical data associated with a possible accident, which will be used to verify the accident conditions and avoid any fraudulent action.

Thus, this document presents the solution developed for the described problem, including the research and evaluation of technologies, the architectural design of the system, the development and the evaluation tests of the solution.

The development of the solution has solved the presented problem, ensuring the safe and immutable storage of the data in a system based essentially on Apache Kafka technology. However, as future work, it is important the optimization of certain components as well as the implementation of additional security measures and evaluation tests in a real scenario.

Keywords: Safety and Health at Work, Accidents at Work, Blockchain, Distributed Systems, Immutable Storage, Apache Kafka, Encryption

Agradecimentos

Em primeiro lugar gostaria de agradecer ao ISEP por todo o conhecimento transmitido ao longo da licenciatura e mestrado, e principalmente ao meu orientador, professor António Costa, por todo o apoio, disponibilidade e aconselhamento em todas as etapas desta dissertação.

Quero também agradecer à Ábaco Consultores pelo projeto desafiante e por toda a disponibilidade ao longo do mesmo. Um especial agradecimento ao meu supervisor, Fernando Cavaleiro, por todo o apoio e conhecimento transmitido, e ao Pedro Fernandes por todo o acompanhamento ao longo do projeto.

Um agradecimento ao meu colega, Diogo Vigo, que me acompanhou e contribui para este projeto, por todo o companheirismo e motivação. Gostaria também de agradecer a todos os colegas e amigos, por facilitarem todo o percurso académico.

Por último, mas não menos importante, agradeço aos meus pais, por tudo o que me proporcionaram durante a vida e percurso académico. À minha irmã, por todo o apoio e compreensão. E um especial agradecimento à minha namorada, por todo apoio, paciência e compreensão ao longo dos últimos meses.

“If you set your goals ridiculously high and it’s a failure, you will fail above everyone else’s success.”

– James Cameron

Índice

Lista de Figuras

Lista de Tabelas

Lista de Código

Acrónimos e Símbolos

Lista de Acrónimos relevantes

AES	*Advanced Encryption Standard*
API	*Application Programming Interface*
BFT	*Byzantine Fault Tolerant*
CTR	*Counter*
DB	Data Base
FT	*Fault Tolerant*
IoT	*Internet of things*
JSON	*JavaScript Object Notation*
JWT	*JSON Web Tokens*
KSQL	*Kafka streaming SQL engine*
PoW	*Proof of work*
REST	*Representational State Transfer*
RDER	Registo de dados de exposição a risco
RGPD	Regulamento Geral de Proteção de Dados
SST	Segurança e Saúde no Trabalho
SQL	*Structured Query Language*
ZAB	*Zookeeper Atomic Broadcast*

1 Introdução

O presente documento foi desenvolvido no âmbito do Mestrado de Engenharia de Software, do departamento de Engenharia Informática do Instituto Português de Engenharia do Porto (ISEP), tendo como objetivo final a atribuição do grau de mestre em Engenharia Informática. A dissertação apresentada foi desenvolvida em contexto empresarial, tendo como objetivo solucionar um problema proposto pela empresa Ábaco Consultores, aplicando essa solução a um produto já comercializado pela empresa, o *inCloud for Safemed*. A presente dissertação será ainda utilizada como suporte para a integração de soluções idênticas em projetos futuros.

1.1 *inCloud for Safemed*

O *inCloud for Safemed* [2] é um produto destinado à área de SST (Segurança e Saúde no trabalho), desenvolvido pelo departamento *EDGE Solutions*, da Ábaco Consultores. O produto pretende facilitar a gestão operacional, gestão administrativa e gestão clínica das empresas que prestam serviços de SST, bem como das empresas que fazem a gestão interna desses mesmos processos.

1.2 Contexto

Dados publicados pela *International Labour Organization* (*ILO*) em 2017 [1] mostram que foram registados 374 milhões de casos de acidentes de trabalho por ano e registados 2.78 milhões de casos mortais devido a acidentes de trabalho.

De acordo com os dados apresentados, é notável que, apesar da consciencialização para a importância da STT e dos avanços significativos na prevenção de acidentes de trabalho, exista ainda um número considerável de acidentes de trabalho. Além disso, as empresas prestadoras de serviços de seguros de acidentes de trabalho necessitam de dados fiáveis que confirmem a ocorrência do sinistro, só assim poderão prestar o seu serviço, assegurando-se que não se trata de um ato fraudulento.

1.3 Problema

O problema pode ser dividido em duas partes, a primeira parte diz respeito ao número de acidentes de trabalho, sejam eles mortais ou não, e a segunda parte consiste em todos os processos pós acidente. A solução a ambas as partes foram desenvolvidas em conjunto uma vez que se complementam para dar resposta ao problema.

Assim, a primeira parte do problema, desenvolvida pelo meu colega Diogo Vigo (nº mecanográfico 1140397), consiste em controlar a ocorrência acidentes de trabalho, isto é, a verificação e registo de situações que possam ter contribuído para que o sinistro ocorresse, e a análise dessas situações de forma a prever a ocorrência de futuros acidentes.

Todo o processo pós acidente pode ser bastante complexo e demorado, uma vez que o sinistro deve ser analisado e validado por várias entidades até que seja declarado como acidente de trabalho. Assim, a segunda parte do problema, apresentada neste documento, recai sobre o processo de confirmação do sinistro, no qual é necessário assegurar que o mesmo ocorreu e que não se trata de qualquer tipo de fraude.

De acordo com o apresentado, o problema ao qual este documento se refere baseia-se essencialmente numa questão “Como comprovar e facilitar todo o processo pós acidente?”.

1.4 Objetivos e abordagem proposta

O objetivo passa por criar uma solução que consiga obter e armazenar dados de colaboradores em tempo real, verificar qualquer irregularidade presente nos mesmos e registar essas ocorrências. Assim, é possível o controlo em tempo real dos dados dos colaboradores e a verificação de ocorrências irregulares, que poderão ser utilizadas para comprovar a existência de qualquer acidente ocorrido, ou para provar a inculpabilidade da empresa num caso em que a mesma seja acusada de negligência em relação a determinado acidente de trabalho.

Para ambas as áreas da solução foi preconizada uma abordagem pela empresa, sendo deste modo possível estruturar a solução de forma conceptual com base nas abordagens propostas, como é apresentado na Figura 1.

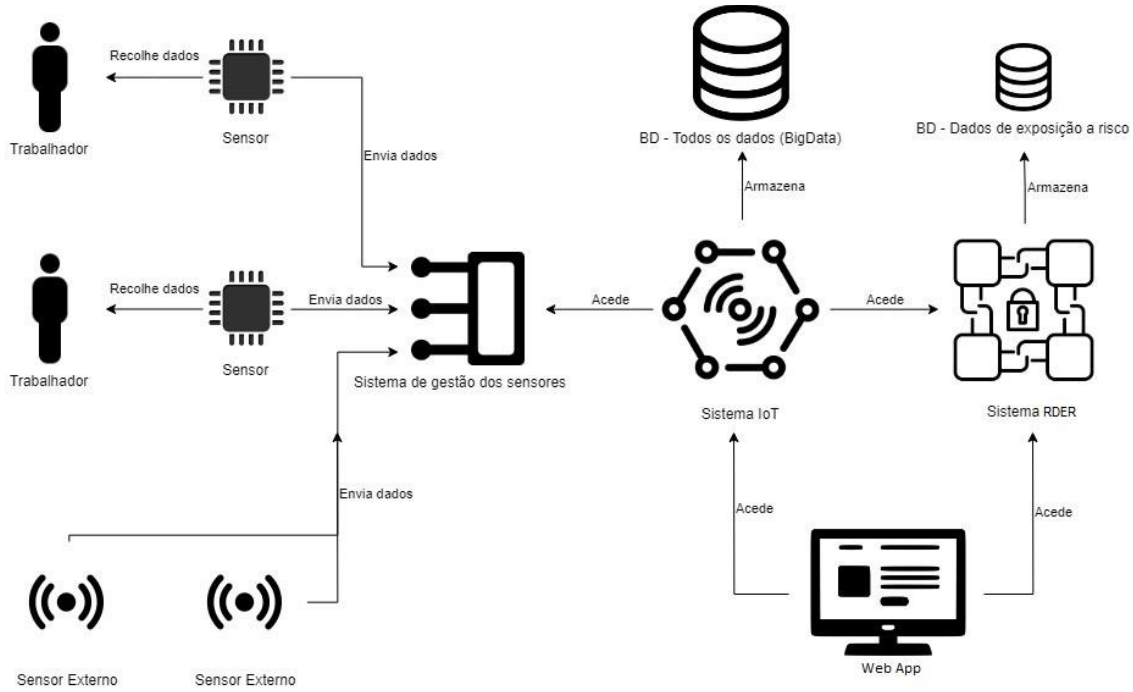


Figura 1: Diagrama conceptual da solução

De acordo com o diagrama apresentado e as áreas referidas anteriormente, é possível descrever a solução proposta da seguinte forma:

- ***IoT*** – A solução proposta para esta área baseia-se no conceito de *IoT* (*Internet of Things*) e *Big Data* e trata de todo o processo de recolha de dados dos colaboradores e dos espaços do edifício da empresa, para isso utilizam-se sensores para a recolha de vários tipos de dados;
- **RDER (Registo de dados de exposição a risco)** – A solução proposta para esta área, que é descrita durante o presente documento, corresponde ao processo de armazenamento e gestão dos valores críticos (dados de exposição a risco) captados pelo sistema *IoT*. Os valores dados como críticos deverão ser armazenados de forma imutável e com as devidas medidas de segurança para que se mantenham fidedignos, para tal foi proposta a utilização de tecnologias *Blockchain*, uma vez que apresentam funcionalidades idênticas às requeridas. O sistema *RDER* deverá ser implementado como uma camada de abstração para o registo de qualquer informação, permitindo, assim, a fácil integração do sistema em produtos futuros, independentemente do modelo de negócio.

Desta forma, dada a estrutura modular e abstrata dos sistemas desenvolvidos, o produto *InCloud for Safemed* e outros produtos da empresa poderão integrar as funcionalidades dos mesmos, operando assim de forma totalmente independente.

1.5 Resultados Esperados

Com o desenvolvimento deste projeto é espectável que a solução permita a total integração no produto da empresa, o *InCloud for Safemed*, disponibilizando assim as funcionalidades previstas inicialmente, como o registo dos dados de exposição ao risco, complementando assim o produto com funcionalidades ricas na área de SST.

A solução final deverá também estar preparada para ser implementada sobre outros produtos da empresa, independentemente do modelo de negócio do mesmo, podendo também ser apresentada com um serviço único e independente que pode ser utilizado pelo cliente como o mesmo bem entender.

É esperado que o sistema desenvolvido cumpra da melhor forma as suas funcionalidades, permitindo o registo dos dados críticos de forma imutável, segura e eficiente, garantindo a fiabilidade desses dados a longo prazo.

1.6 Análise de valor

Apesar da utilização do sistema RDER ser limitada a um número restrito de casos de uso, no caso de existir um produto em que realmente seja vantajoso, o número de recursos e custos associados ao uso do mesmo podem ser equilibrados com as vantagens que este sistema tem para oferecer.

Assim, os clientes poderão usufruir de produtos mais completos da empresa, quando integrado com o sistema RDER, dadas as suas características de registo de dados relativamente aos sistemas tradicionais de armazenamento de informação. Estas características, que são fatores diferenciadores relativamente a outros produtos existentes no mercado, oferecem o valor que o cliente procura no produto.

1.7 Estrutura do Documento

O presente documento apresenta-se estruturado em oito capítulos. No **primeiro capítulo** é apresentado um enquadramento do projeto e do produto no qual será integrado, é descrito o contexto e o problema, os objetivos, a abordagem proposta e os resultados esperados, por fim é realizada uma introdução à análise de valor.

No **segundo capítulo** é apresentado em pormenor o contexto e o problema e, descrita detalhadamente a análise de valor.

No **terceiro capítulo** é efetuada uma introdução teórica aos conceitos fundamentais do *Blockchain* e às indústrias em que pode ser aplicado, são descritas as tecnologias analisadas, é apresentada a abordagem ao RGPD (Regulamento Geral de Proteção de Dados) para o problema em questão e por fim apresentadas algumas soluções existentes.

No **quarto capítulo** são apresentados os requisitos da solução idealizada, são avaliadas detalhadamente as tecnologias apresentadas no capítulo anterior e feita uma comparação entre as mesmas.

No **quinto capítulo** é realizada uma avaliação entre as várias soluções ao problema, é apresentado em detalhe o *design* da solução mais indicada e posteriormente, de forma mais genérica, o *design* das soluções alternativas.

No **sexto capítulo** é descrito todo o processo de desenvolvimento da solução.

No **sétimo capítulo** é efetuada a avaliação e experimentação da solução desenvolvida.

No **oitavo capítulo** são apresentadas as conclusões relativas à solução apresentada no documento, incluindo os objetivos alcançados, as limitações e o trabalho futuro.

2 Contexto

No presente capítulo é descrito em detalhe o contexto e o problema em análise, é também apresentada toda a análise de valor da solução, desde o seu enquadramento através do modelo de NCD (*New Concept Development Model*) até ao valor da solução para os utilizadores.

2.1 Contexto e Problema

Como é apresentado na secção 1.2, o número de acidentes de trabalho a nível mundial é exposto como uma situação problemática na área da SST, mesmo com toda a consciencialização sobre a importância da segurança e saúde no trabalho. Apesar dos dados referidos anteriormente corresponderam a uma escala mundial, Portugal apresenta dados estatísticos alarmantes relativamente a outros países da união europeia.

Uma publicação estatística feita pela *Eurostat* em Junho de 2018 [3] mostra o gráfico de acidentes fatais no trabalho registados na Europa em 2015, em que Portugal foi representado na 4ª posição com um valor de 3.54 trabalhadores mortos em 100 000 acidentes registados, como é apresentado na Figura 2.

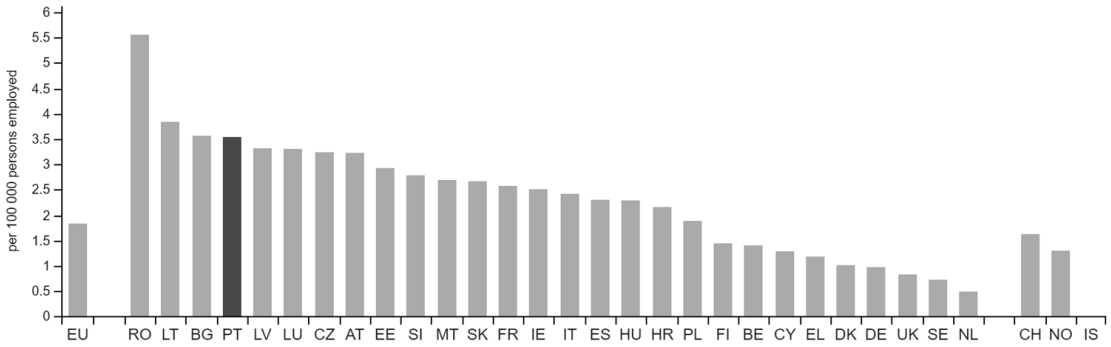


Figura 2: Acidentes de trabalho fatais em 2015. Fonte: [3]

O Gabinete de Estratégia e Planeamento (GEP) disponibilizou no dia 23 de Outubro de 2018 relatórios estatísticos sobre acidentes de trabalho em Portugal no ano de 2016 [4], tendo sido registados 207567 casos de acidentes de trabalho, em que 138 foram registados como mortais.

Dados de 2014 e 2015 do mesmo relatório mostram valores bastante idênticos de acidentes de trabalho por ano, registando um valor médio de 206 524 acidentes ao longo dos últimos três anos, como é representado na Figura 3.

INFORMAÇÃO GERAL (por ano)	2014	2015	2016
Acidentes de trabalho (Total/Mortais)	203 548 / 160	208 457 / 161	207 567 / 138
Taxa de incidência (Total/Mortais)	4 523,8 / 3,6	4 582,8 / 3,5	4 507,2 / 3,0
Total de AT com DTP	137 345	142 031	142 647
Total DTP	5 324 131	5 459 744	5 333 835
Média de DTP por AT	38,8	38,4	37,4

Figura 3: Informação geral de acidentes de trabalho por ano. Fonte: [4]

Apesar de qualquer setor de atividade estar suscetível a acidentes de trabalhos, existem setores onde a probabilidade de ocorrência de acidentes é superior, seja por envolver o manuseamento de máquinas ou por existir contacto com materiais perigosos. Dessa forma, setores mais suscetíveis a acidentes deverão estar melhor preparados para qualquer acontecimento desse tipo, seja para prevenir ou para tratar dos processos posteriores ao acidente.

Independentemente do setor ou da gravidade acidente, pode ser registado como acidente de trabalho qualquer situação que no local e tempo de trabalho cause direta ou indiretamente lesão corporal, perturbação funcional ou doença de que resulte redução da capacidade de trabalho ou de ganho, ou ainda a morte (artigo 8.º da Lei n.º 98/2009, de 4 de setembro) [5], sendo que todos os trabalhadores tem direito a assistência e justa reparação, quando vítimas de acidente de trabalho ou de doença profissional (Art. 59.º da Constituição da República Portuguesa) [6].

Após a ocorrência de um acidente de trabalho, o empregador é responsável pela investigação do mesmo (Lei n.º 102/2009, de 10 de Setembro) [7], que consiste em apurar as causas que contribuíram, direta ou indiretamente, para a sua ocorrência. O empregador deverá comunicar o sinistro à ACT (Autoridade para as Condições do Trabalho) num prazo de 24 horas [8], no qual deverá ser enviado um formulário com a informação necessária para a confirmação do mesmo. Deverá também entrar em contacto com a empresa prestadora dos serviços de seguro para comprovar a ocorrência do sinistro, garantindo que não se trata de um ato fraudulento.

No caso de qualquer trabalhador, vítima de acidente de trabalho, acusar a empresa de negligência, o empregador necessita de provar que todas as normas de segurança foram cumpridas e que não existiu alguma falha que pudesse ter causado o sinistro. [9]

Com base nos dados apresentados, é notável a necessidade de criar soluções que permitam controlar e até reduzir o número de acidentes de trabalho, e permitam facilitar todo o processo pós acidente, como a comprovação do sinistro a seguradoras.

2.2 Análise de Valor

Análise de valor é baseada no modelo *NCD* (*New Concept Development*), em vários conceitos de valor, na proposta de valor apresentada e por fim no modelo de negócio *CANVAS*.

2.2.1 *New Concept Development Model*

O modelo *NCD* [10] foi criado como forma de representação da primeira parte do processo de *Front-End of Inovation*. Este processo é ainda composto por mais duas fases, o desenvolvimento de novo produto (*NPD*) e comercialização.

O *NCD* é constituído por três componentes, o motor, os fatores de influência e os cinco elementos internos. O modelo é representado na Figura 4 num diagrama em forma circular, mostrando assim a ligação entre os componentes do mesmo, sendo que o motor representa os níveis superiores de gestão de uma organização responsável por gerir os elementos do modelo, e os fatores influenciadores representam "o quê" e "quem" influencia o processo de inovação, fatores que podem ser tanto internos como externos. Por fim, os cinco elementos do modelo representam a identificação da oportunidade, a análise de oportunidade, a geração e enriquecimento de ideias, a seleção de ideias e a definição do conceito.

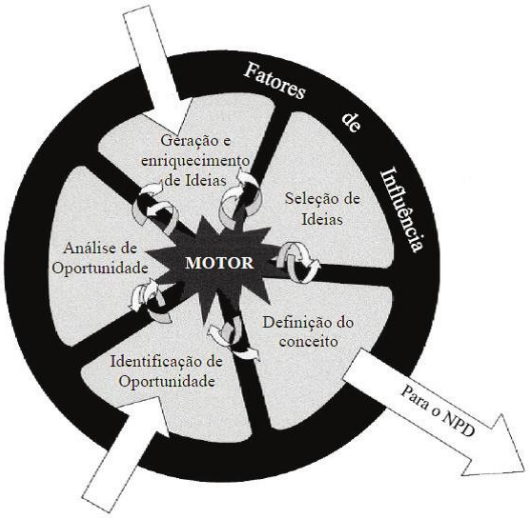


Figura 4: *New Concept Development Model* (NCD). Fonte: [10]

2.2.1.1 Identificação da oportunidade

A oportunidade surgiu através de um dos produtos da empresa já presente no mercado, o *inCloud for Safemed*, que se insere no mercado de SST e no qual existiu a necessidade de

desenvolver uma solução que garantisse o armazenamento, confidencialidade, integridade e fiabilidade de dados críticos captados pelos sensores do sistema *IoT*, sendo o *Blockchain* o conceito preconizado para resolver este problema.

Desta forma é possível utilizar os dados registados para complementar situações de acidentes de trabalho e, dado o elevado número de acidentes de trabalho e a necessidade de garantir a fiabilidade de informação para que o acidente seja validado por empresas prestadoras de serviços de seguros, verifica-se a oportunidade de integrar esta solução no produto da empresa.

2.2.1.2 Análise da oportunidade

O produto *inCloud for Safemed* já se encontra incorporado em várias empresas que prestam serviços de SST e empresas de grande dimensão que possuem o seu próprio departamento de SST, tanto empresas Nacionais como Internacionais. Com a integração do Sistema RDER em conjunto com o sistema *IoT* é possível criar um produto mais completo e robusto e, deste modo atingir clientes que antes não eram possíveis. Através do desenvolvimento desta solução, e dada a ideia inicial de um sistema abstrato, será possível integrar o sistema noutros produtos, independentemente do modelo de negócio e assim completar outros produtos da empresa.

2.2.1.3 Geração e enriquecimento de ideias

Foram realizadas reuniões semanais juntamente com os intervenientes do projeto, entre eles o supervisor da tese na empresa, um colaborador da empresa e o Diogo Vigo, responsável pelo desenvolvimento de outra tese e simultaneamente do sistema *IoT*. As reuniões foram utilizadas para discussão de ideias relativas aos projetos em desenvolvimento, a forma como os sistemas *IoT* e RDER se complementam, como são integrados no *inCloud for Safemed* e como são estruturados para serem facilmente integrados em produtos futuros. Foram também discutidas questões mais detalhadas, como as tecnologias a utilizar, a arquitetura dos sistemas e questões relativas ao RGPD.

2.2.1.4 Seleção de ideias

Como resultado das reuniões foi surgindo a solução ideal à medida que as várias soluções eram discutidas e comparadas, sendo que a cada reunião o sistema foi modulado para que correspondesse aos requisitos pretendidos, mas sempre respeitando as limitações apresentadas pela empresa, pelo mercado/clientes e pelo RGPD.

2.2.1.5 Definição do conceito

O conceito baseia-se num sistema seguro e abstrato que armazene dados num formato imutável, na medida em que o sistema seja totalmente independente do modelo de negócio do produto que o utiliza e que toda a informação registada se mantenha íntegra, de forma a ser considerada fiável. O sistema deve possuir mecanismos para que essa informação, além de ser registada, possa ser facilmente acedida pelas aplicações com as devidas permissões de acesso.

No caso do produto *inCloud for Safemed*, o sistema *IoT* deverá recolher dados dos colaboradores e ambientais e analisar os mesmos de forma a verificar irregularidades, dados irregulares devem também ser registados no sistema RDER, uma vez que se pode tratar de

informação essencial para determinado sinistro. O próprio sistema *inCloud for Safemed* possuirá um mecanismo de análise dos dados dos sensores do sistema *IoT* conforme os dados médicos de cada trabalhador, o que permitirá a análise de dados críticos, e no caso de serem confirmados como tal, esses dados devem ser registados no sistema RDER.

2.2.2 Valor, valor percecionado e valor para o cliente

Os vários conceitos de valor podem ser abordados de formas distintas, sendo necessário interpretar cada um deles e simultaneamente o modo como são apresentados pela solução.

2.2.2.1 Valor

"*Value creation is a concept that is difficult to achieve, understand, model and/or conceptualize. Some authors consider value creation a trade-off between benefits and sacrifices perceived by customers during a supplier's offering*" [11]

O conceito de valor é bastante complexo e pode ser abordado de formas diferentes, porém em todos as situações deve existir o esforço para que seja atingido o equilíbrio entre os benefícios e sacrifícios do cliente.

A solução apresentada no documento pretende complementar os produtos da empresa com um módulo seguro e fiável para armazenamento de dados, sendo que o cliente beneficia diretamente desta funcionalidade sobre os seus dados, contudo a solução pode exigir algum sacrifício de desempenho e custos associados.

2.2.2.2 Valor percecionado

"*Different customers perceive different value for the same products/services. In addition, organizations involved in the purchasing process can have different perceptions of customers' value delivery*" [12]

O valor percecionado varia de acordo com cada indivíduo, desta forma o cliente e a empresa que vende o produto podem atribuir diferente valor a determinado produto, sendo necessário ambas as partes se compreenderem e equilibrarem o valor atribuído por cada um.

Assim, o valor atribuído por parte da Ábaco sobre o sistema RDER baseia-se no facto de ser possível a integração do mesmo em qualquer produto da empresa, aperfeiçoando assim os seus produtos e desta forma chegar a clientes e mercados que de outra forma não eram atingíveis. Por parte do cliente, o valor é representado conforme as necessidades do negócio, na medida em que para o caso de ser necessário o armazenamento de dados de forma segura, o sistema RDER corresponderá na íntegra a esse requisito.

2.2.2.3 Valor para o cliente

"*Value for the customer (VC) is any demand-side, personal perception of advantage arising out of a customer's association with an organization's offering, and can occur as reduction in sacrifice; presence of benefit (perceived as either attributes or outcomes); the resultant of any*

weighed combination of sacrifice and benefit; or an aggregation, over time, of any or all these" [13]

O valor do cliente é representado pelo equilíbrio entre os benefícios e sacrifícios que um produto pode apresentar, desta forma um produto é valorizado quando são apresentados benefícios que se destaquem consideravelmente sobre os sacrifícios.

Assim, a integração do sistema RDER como módulo de outro produto pode ser representada para o Cliente pelos benefícios e sacrifícios da Tabela 1.

Tabela 1: Benefícios e sacrifícios para o Cliente

Benefícios	**Sacrifícios**
Segurança dos dados	Desempenho
Integridade dos dados	Elevados custos
Independente do modelo de negócio	

De forma mais detalhada, os benefícios e sacrifícios da Tabela 1 podem ser retratados da seguinte forma:

- **Benefícios:**
  - Segurança dos dados – Os dados registados são encriptados e o acesso aos mesmos é restrito;
  - Integridade dos dados – Os dados são imutáveis, não existindo assim possibilidade de serem alterados ou eliminados;
  - Independente do modelo de negócio – O sistema é completamente abstrato, sendo que desta forma é possível integrar o mesmo em qualquer produto ou utilizar o mesmo de forma independente.
- **Sacrifícios:**
  - Desempenho – O sistema, devido a todos os mecanismos de segurança apresentados, sofre uma queda no tempo de resposta no registo e acesso de dados, tempo de resposta esse que diminuirá à medida que o volume de dados armazenados aumenta;
  - Elevados custos – Para que o sistema se comporte com um desempenho aceitável, o cliente tem de suportar os elevados custos de funcionamento do mesmo.

2.2.3 Proposta de valor

Com o desenvolvimento da solução no presente documento, e dado o nível de abstração da solução a desenvolver, será possível a venda do sistema RDER como módulo extra em produtos da empresa ou como um módulo único e independente.

Desta forma a empresa beneficiará pelo facto da solução desenvolvida permitir a robustez dos seus produtos e ser característica distinguível em relação a outros produtos do mercado,

conseguindo assim atingir um leque mais abrangente de clientes. Isto permitirá à empresa crescer e até desenvolver produtos para novos mercados.

O cliente, no caso de já ser portador de produtos da empresa, terá a possibilidade de comprar o serviço como módulo externo se assim o entender, no caso de novos clientes, terão à sua disponibilidade produtos ainda mais completos.

Apesar de a utilização deste tipo de soluções ser restrita a determinados negócios, dado que nem todos os negócios necessitam do nível de segurança e imutabilidade oferecido pelo sistema RDER, a sua integração em negócios que o necessitem é de grande valor para o cliente.

2.2.4 Modelo de negócio *CANVAS*

O modelo de negócio *CANVAS* é utilizado como representação estruturada da forma como se pretende criar e entregar valor para o cliente, sendo estruturado em nove blocos:

- **Segmento de clientes** – Perfil dos clientes alvo a quem se favorece valor, as suas necessidades e problemas;
- **Proposta de valor** – O que se oferece aos clientes alvo e fatores diferenciadores em relação a outros produtos do mercado;
- **Canais** – Forma como os produtos e serviços vão chegar aos clientes;
- **Relação com os clientes** – Estratégias para fidelizar e conquistar os clientes;
- **Receitas** – Valor monetário definido para o produto e serviço e outras formas de obtenção de lucro;
- **Recursos-chave** – Descrição dos recursos humanos e financeiros;
- **Atividades-chave** – Discriminação das atividades e processos necessários para que os produtos e serviços cheguem com sucesso ao cliente;
- **Parceiros-chave** – Descrição dos fornecedores da empresa favoráveis ao produto;
- **Custos** – Custos financeiros associados a todos os processos de desenvolvimento e manutenção do produto.

A Figura 5 representa o modelo *CANVAS* preenchido pela ordem dos blocos apresentados anteriormente.

<table>
<tr>
<td rowspan="2">Parceiros-chave

• Parceiros interessados em suportar parte dos custos do sistema, no caso do inCloud for Safemed, os parceiros interessados seriam a ACT (Autoridade para as condições de Trabalho) e a DGS (Direção-Geral da Saúde).</td>
<td>Atividades-chave

• Desenvolvimento e manutenção do produto;
• Garantir a implementação do sistema nos vários servidores.</td>
<td rowspan="2">Proposta de Valor

• É oferecida a garantia que os dados são armazenados e que existe um nível de segurança elevado sobre esses dados, bastante superior às tecnologias de armazenamento convencionais;
• Dada a flexibilidade em integrar a solução em vários produtos, qualquer produto que a empresa oferece terá o armazenamento seguro da informação como fator diferenciador no mercado.</td>
<td>Relação com os clientes

• No caso de já serem clientes da empresa o mesmo poderá adicionar o módulo extra ao produto;
• No caso de não serem clientes, o fator diferenciador dos produtos é essencial para os tornar clientes.</td>
<td rowspan="2">Segmento de clientes

• Clientes atuais da empresa que pretendam um produto mais completo, como é o caso do inCloud for Safemed;
• Empresa que necessitam de armazenar dados de forma segura e manter a integridade da informação registada.</td>
</tr>
<tr>
<td>Recursos-chave

• Recursos humanos;
• Escritórios e salas para servidores;
• Serviços cloud;</td>
<td>Canais

• No caso de se tratar de um módulo extra de um produto, a sua distribuição será feita conforme o produto em questão;
• No caso de ser um serviço único a distribuição poderá ser feita via cloud.</td>
</tr>
<tr>
<td colspan="3">Custos

• Custo de manutenção dos servidores nos quais o sistema está implementado.</td>
<td colspan="3">Receitas

• Licenciamento da solução;
• Subscrição e pagamento de mensalidade para uso da solução.</td>
</tr>
</table>

Figura 5: Modelo de negócio *CANVAS*

3 Estado da Arte

No presente capítulo é apresentada toda a pesquisa realizada para solucionar o problema apresentado neste documento. Inicialmente, e de acordo com a abordagem inicialmente proposta, é apresentada uma introdução aos conceitos base do *Blockchain*, à sua aplicabilidade nos diversos setores e às tecnologias existentes. São apresentadas de seguida tecnologias de bases de dados e o *Apache Kafka*, são expostos temas relativos à encriptação e a legislação do RGPD e por fim apresentadas as soluções já existentes.

3.1 *Blockchain*

A tecnologia *Blockchain* foi introduzida por Satoshi Nakamoto em 2008, com a invenção da criptomoeda *Bitcoin* [14], idealizada como um sistema *peer-to-peer* que permitisse pagamentos *online* seguros sem qualquer intervenção de instituições financeiras.

3.1.1 *Ledger* distribuído

Compreende-se por *ledger* distribuído uma base de dados de registos encriptados partilhada por vários nós da rede sem nenhuma autoridade central, ao contrário dos sistemas centralizados e descentralizados (Figura 6). Todos os participantes da rede têm uma cópia idêntica do *ledger* e qualquer alteração ao *ledger* é refletida em todas as cópias da rede em minutos ou, em alguns casos, em segundos. [15]

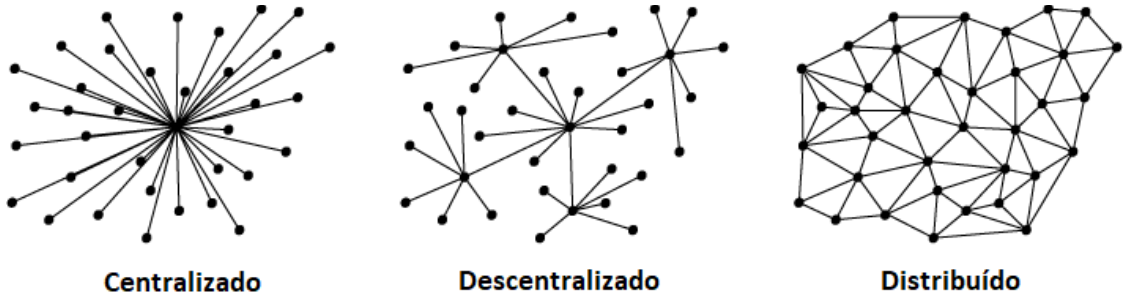


Figura 6: Comparação de sistemas centralizados, descentralizados e distribuídos.

Para manter o nível de segurança de uma arquitetura distribuída, como o *Blockchain*, qualquer alteração ao *ledger* tem de ser validada e aceite pelos nós da rede através de um mecanismo de consenso, descrito na secção 3.1.3.

3.1.2 *Blockchains* Públicas, Privadas e Híbridas

Cada modelo de negócio é diferente e dessa forma foi necessário desenvolver tecnologias *Blockchain* com diferentes abordagens que se adequassem às diferentes indústrias. Após a primeira implementação do *Blockchain*, numa rede pública, foram aparecendo novas abordagens à tecnologia com diferentes restrições de acesso, aparecendo as *Blockchains* privadas e híbridas. [16]

- ***Blockchains* Públicas** – Como o nome sugere, uma *Blockchain* pública é totalmente aberta ao público, permitindo que qualquer pessoa possa participar na rede. Normalmente utiliza mecanismos de incentivo aos participantes, facilitando assim o crescimento da mesma, como é o caso do *Bitcoin*;
- ***Blockchains* Privadas** – Uma *Blockchain* privada ao contrário da *Blockchain* pública, restringe o acesso à rede apenas a participantes autorizados. Permite desta forma ter um maior controlo da rede, definindo quem tem acesso à escrita ou leitura de dados. Possuem uma abordagem mais centralizada e normalmente são mais eficientes;
- ***Blockchains* Híbridas** – Conhecidas também como *Blockchains* de Consórcio, são parcialmente privadas e relativamente recentes. Ao contrário das *Blockchains* privadas, que são controladas por uma única organização, as *Blockchains* Híbridas são controladas por um consórcio de organizações, descentralizando assim determinadas decisões na rede.

3.1.3 Algoritmo de consenso

Como foi descrito anteriormente, num sistema distribuído como o *Blockchain*, existe a necessidade de aplicar um mecanismo de consenso para manter o nível de segurança da rede. Assim sendo, o algoritmo de consenso, consiste num mecanismo de tolerância à falha necessário para validar a introdução ou alteração de dados num sistema multiagente. [17]

Desde a introdução do *Blockchain* já foram desenvolvidos vários algoritmos de consenso, alguns deles como melhoria a outros já existentes e outros com abordagens completamente distintas. De seguida estão representados alguns dos algoritmos mais populares:

- ***Proof of Work* (*PoW*)** – Processo conhecido por *mining*, que consiste no cálculo de uma *hash* para introduzir um novo bloco. Introduzido pelo *Bitcoin* em 2008 [14] e atualmente utilizado por várias criptomoedas, como o *Ethereum*;
- ***Proof of Stake* (*PoS*)** – Desenhado para corrigir o problema de alto consumo de energia do *PoW*, um *miner* tem de apostar no bloco válido para ter o privilégio de gerar um novo bloco. Assim, quanto maior a quantidade de blocos de um participante, maior é a probabilidade de adicionar um novo bloco na rede; [18]
- ***Delegated Proof of Stake* (*DPoS*)** – Este algoritmo, desenvolvido por Dan Larimer em 2014, apesar de idêntico ao *PoS*, introduz um sistema em tempo real de votação de "super representantes" que ficarão responsáveis por validar novos blocos. Destacando-se assim por ser mais rápido, justo e democrático, uma vez que exige a participação da comunidade. [19] Já implementado por criptomoedas como EOS, *Lisk*, *Steem*, etc;
- ***Proof of Authority* (*PoA*)** – Consiste num mecanismo no qual as transações são validadas apenas por um grupo de administradores e, apesar de tornar o processamento de transações mais rápido, foi desenvolvido para *Blockchains* privadas, uma vez que a decisão sobre as transações estão centralizadas num grupo de administradores; [20]
- ***Proof of Elapsed Time* (*PoET*)** – Desenvolvido pela Intel e em parceria com o projeto *HyperLedger* da *Linux Foundation*, este algoritmo de consenso é utilizado pela tecnologia Intel *SawtoothLake* e utiliza um sistema de "lotaria justa"; [21]
- ***Practical Byzantine Fault Tolerance* (*PBFT*)** – Desenvolvido em 1999 no IMT [22], pretende solucionar o problema *Byzantine Generals* com um algoritmo no qual cada nó decide se determinada informação deve ser adicionada ou não, e após decidir comunicar a outro nó, este processo sucedesse várias vezes até existir um consenso entre a maioria; [22]

- ***Directed Acyclic Graph* (*DAG*)** – Uma nova abordagem à estrutura do *Blockchain* através de um grafo orientado que utiliza ordenação topológica. Desta forma, e ao contrário do *Bitcoin* e outras tecnologias que utilizam *PoW*, o *DAG* permite a introdução de vários blocos em simultâneo, tornando todo o processo muito mais rápido. [23] Atualmente usado em projetos como *IoT Chain*, *IOTA* e *Byteball*.

3.1.4 Imutabilidade

O conceito de imutabilidade é alcançado na tecnologia *Blockchain* através da combinação dos conceitos apresentados nas secções seguintes, explicados com base na tecnologia *Bitcoin*. Tal permite assegurar que dados registados no sistema não possam ser alterados ou eliminados.

3.1.4.1 Blocos

A Figura 7 representa a estrutura da cadeia de blocos, apresentando a forma como os blocos estão interligados entre si e de que forma as transações e outros dados são registados.

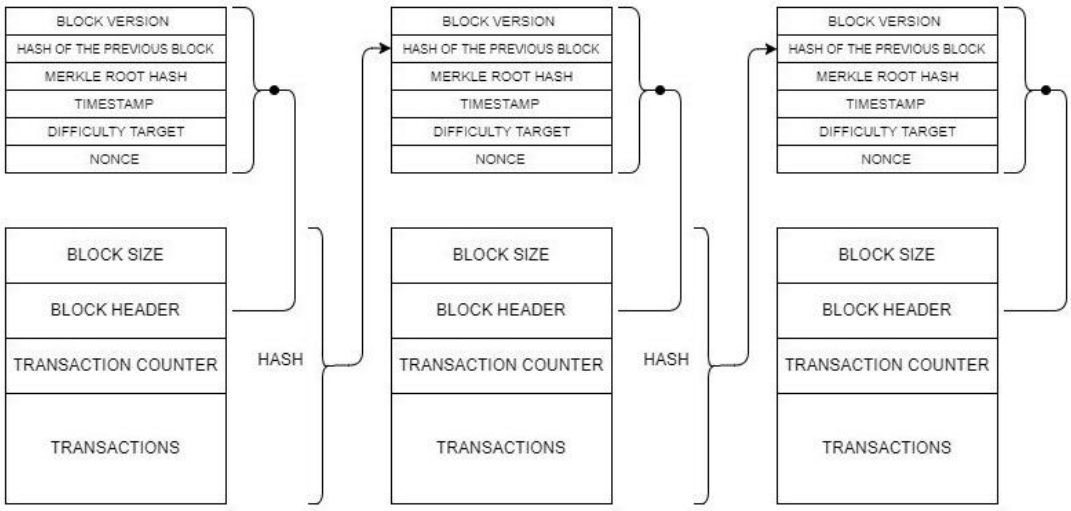


Figura 7: Estrutura detalhada do bloco no *Bitcoin*. Fonte: [24]

Para uma melhor perceção da estrutura do bloco, são brevemente descritos as propriedades de maior relevância registadas em cada bloco [25]:

- ***Hash* do próprio bloco** – Cada bloco tem a sua própria *hash*, gerada de acordo com a informação contida nesse mesmo bloco;
- ***Hash* do bloco anterior** – Referência ao bloco anterior através da *hash* do mesmo;
- ***Timestamp*** – Representa temporalmente a criação do bloco;

- ***Merkle Root Hash*** – Referência em formato de *hash* à *Merkle Tree* (contém o conjunto de transações confirmados por este bloco);
- **Transações** – Conjunto de transações confirmados por este bloco;
- ***Difficulty Target* (nBits)** – *Hash* de referência à dificuldade em gerar um novo bloco. O valor da dificuldade é atualizado a cada 2016 blocos ou a cada 2 semanas;
- ***Nonce*** – Número incrementável de 32 bits utilizado no processo de *mining* para descobrir uma *hash* válida.

3.1.4.2 *Hashing*

O conceito de *hashing* baseia-se na encriptação de informação através de um algoritmo de encriptação, gerando assim uma chave única para a informação inicial. No caso do *Bitcoin*, as transações da *Merkle Tree* e blocos são encriptadas recorrendo ao algoritmo *SHA-256*, resultando uma chave única de tamanho fixo independentemente do tamanho do *input* [26]. Na Tabela 2 é possível observar um exemplo prático do conceito descrito.

Tabela 2: Exemplo prático do algoritmo de *hashing SHA-256*

Input	***SHA-256 Hash***
Hash	A91069147F9BD9245CDACAEF8EAD4C3578ED44F179D7EB6BD4690E62BA4658F2
Blockchain	625DA44E4EAF58D61CF048D168AA6F5E492DEA166D8BB54EC06C30DE07*DB*57E1
Ábaco	657F21B4457906BEF5B13A41A98524DE862228BB32A1AF317C48B90B4153DC3A

Este conceito fornece várias propriedades relevantes para a segurança da tecnologia *Blockchain*, entre as quais:

- **Determinístico** – Independentemente do número de vezes que for introduzido determinado *input*, a função *hash* vai sempre retornar o mesmo resultado;
- **Rápido processamento computacional** – A função *hash* deve estar suficientemente otimizada para calcular a *hash* rapidamente, de outra forma o sistema não seria eficiente dada a quantidade de vezes que essa função é utilizada;
- **Resistência à pré-imagem** – Significa que é computacionalmente difícil ou impossível de determinar o *input* através da *hash*;
- **Qualquer mudança no *input* altera a *hash*** – Uma das funcionalidades críticas para o conceito de imutabilidade no *Blockchain*, dado que qualquer mudança nos dados de *input* afetam a *hash* resultante;

- **Resistência à colisão** – Significa que é muito pouco provável ou impossível encontrar duas *hashes* iguais geradas a partir de *inputs* diferentes. Nenhuma função *hash* é livre de colisões, mas geralmente demora muito tempo a encontrar um caso em que isso aconteça, desta forma no caso do algoritmo *SHA-256*, é seguro assumir que se *h(A) = h(B)*, então *A = B*;
- ***Puzzle Friendly*** – Determina que é inviável tentar manipular os dados de *input* para que a função *hash* retorne um resultado específico.

3.1.4.3 *Merkle Tree*

Patenteado por Ralph Merkle em 1979, o conceito de *Merkle Tree* representa uma estrutura de dados em formato de árvore, na qual cada "nó não-folha" é uma *hash* dos respetivos "nós filhos" e os "nós folha" (*Data Blocks*) estão definidos no nível mais baixo da estrutura da árvore e são utilizados para o armazenamento dos dados. [27] A estrutura da *Merkle Tree* pode ser representada pela Figura 8.

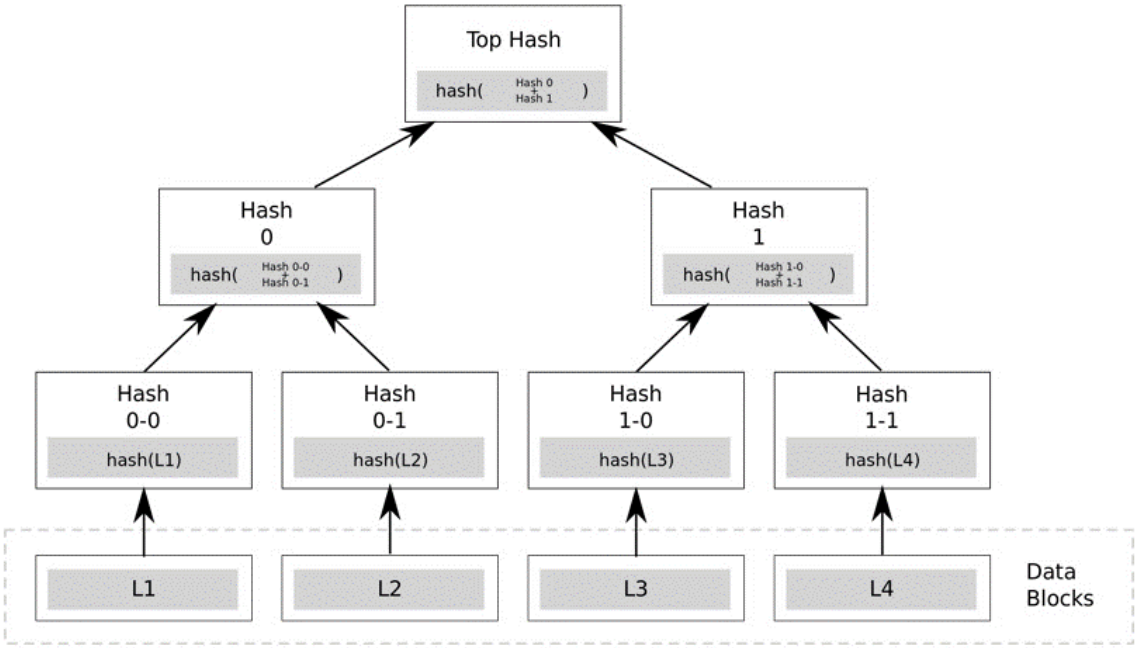


Figura 8: Representação da *Merkle Tree*. Fonte: [27]

Para uma melhor compreensão da Figura 8, *L1* e *L2* representam nós folha, *Hash 0-0* representa um "nó não-folha" em que a sua *hash* é determinada a partir de *L1*, *Hash 0* representa outro "nó não-folha" mas ao contrário do *Hash 0-0*, a sua *hash* é determinada a partir dos nós *Hash 0-0* e *Hash 0-1*. Através desta estrutura toda a árvore é representada pelo nó do nível mais elevado, conhecido por *Root Hash*.

No caso do *Blockchain*, mais especificamente do *Bitcoin*, os *Data Blocks* armazenam os dados das transações de determinado bloco. Desta forma é possível estruturar os dados das

transações, permitindo mapear eficientemente grande quantidades de dados e através do sistema de *hashing* reduzir a quantidade de memória necessária para o armazenamento das transações, uma vez que no *header* do bloco apenas é armazenado o *Root Hash* da *Merkle Tree*.

3.1.4.4 *Mining*

Mining é o processo no qual transações são verificadas e introduzidas no *Blockchain*, no caso do *Bitcoin* é este mesmo processo que permite a adição de novos blocos à cadeira e consequentemente à disponibilização de novos *Bitcoin*s. O processo envolve a compilação de recentes transações em blocos e dessa forma tentar resolver um problema computacionalmente difícil, sendo que o primeiro participante a resolver o problema pode introduzir o novo bloco na cadeia. [28]

O problema consiste em tentar gerar a *hash SHA-256* do *header* do bloco para que o mesmo seja aceite na rede, essa *hash* tem de ser menor ou igual à *hash* definida na *Difficulty Target* (Figura 7). A dificuldade do problema, definida no *header* do bloco como *Difficulty Target (nBit)*, é atualizada a cada 2016 blocos ou a cada 2 semanas. Define-se pelo número de zeros iniciais que a *hash* deve ter, reduzindo drasticamente a probabilidade do cálculo da *hash* e sendo assim necessário incrementar a propriedade *Nonce* (Figura 7) várias vezes até encontrar um *hash* válido. Este processo, como já foi descrito anteriormente é utilizado como *Proof of Work*. [29]

A Tabela 3 descreve um exemplo prático de *mining* no qual é utilizado como *string* base "Hello, world!", e como *Difficulty Target* uma variação da *string* base com 3 zeros iniciais que representa um número de 256 bits com o valor 2^240. As várias variações na *string* consistem na adição do valor do *Nonce* à *string*, que é incrementado a cada tentativa até que seja gerada uma *hash* com um número de 256 bits inferior ao da *Difficulty Target*.

Tabela 3: Exemplo prático de *mining* (*Proof of Work*). Fonte: [30]

String + Nonce	***Hash***	***Hash* (256 bits)**
Hello, world!0	1312af178c253f84028d480a6adc1e25 e81caa44c749ec81976192e2ec934c64	2^252.253458683
Hello, world!1	e9afc424b79e4f6ab42d99c81156d3a1 7228d6e1eef4139be78e948a9332a7d8	2^255.868431117
Hello, world!2	ae37343a357a8297591625e7134cbea 22f5928be8ca2a32aa475cf05fd4266b7	2^255.444730341
…	…	…
Hello, world!4248	6e110d98b388e77e9c6f042ac6b497c ec46660deef75a55ebc7cfdf65cc0b965	2^254.782233115
Hello, world!4249	c004190b822f1669cac8dc37e761cb73 652e7832fb814565702245cf26ebb9e6	2^255.585082774
Hello, world!4250	0000c3af42fc31103f1fdc0151fa747ff 87349a4714df7cc52ea464e12dcd4e9	2^239.61238653

Para este exemplo, são necessárias 4251 tentativas para obter uma *hash* válida, uma vez que quando o valor da *Nonce* é de 4250 a *hash* gerada representa um número de 256 bits (2^239.61238653) inferior ao da *Difficulty Target* definida (2^240).

Em suma, dada a quantidade de poder computacional, tempo e energia gasta neste processo, não se torna viável para um participante malicioso alterar um bloco, uma vez seria necessário refazer o processo de *Proof of Work* para o bloco em questão e para todos os blocos após o mesmo, além de precisar de alcançar e ultrapassar o processo de *mining* dos participantes honestos [14], preenchendo desta forma o conceito de imutabilidade para cado bloco da cadeia.

3.1.5 *Smart Contracts*

Smart Contracts são pequenos programas computacionais armazenados no *ledger* do *Blockchain* e automaticamente executados quando determinadas condições, já pré-definidas, são alcançadas. A implementação de *Smart Contracts* é limitada conforme a tecnologia utilizada, porém, de forma simplificada, são utilizados como intermediário de confiança, certificando-se que determinado acordo entre vários participantes é cumprido. [31]

O conceito foi apresentado por Nick Szabo em 1996, mas apenas foi implementado de forma prática em 2013 por Vitalik Buterin na plataforma *Ethereum*, introduzindo às tecnologias *Blockchain* mais versatilidade e outros benefícios, como:

- **Velocidade e exatidão** – Como é um processo digital e automático, não existe desperdícios de tempo com burocracias, erros humanos e papelada, sendo muito mais eficientes do que os contratos tradicionais;
- **Confiança** – As transações são efetuadas automaticamente conforme as regras definidas no *Smart Contract*, regras essas que são definidas aquando da implementação do mesmo;
- **Segurança** – Uma vez que tanto as transações como o código do *Smart Contract* estão presentes na *Blockchain*, herda o conceito de imutabilidade descrito na secção 3.1.4;
- **Poupança** – O conceito de poupança refere-se ao facto de não ser necessário intermediários externos, nem nenhum participante para validar os termos do acordo.

3.1.6 *Blockchain* nas várias Indústrias

Compreendido o conceito da tecnologia, é possível extrair os benefícios essenciais da mesma e perceber de que forma esses benefícios se enquadram nas diferentes indústrias. Entre os vários benefícios do *Blockchain*, os de maior relevância são:

- Descentralização;
- Segurança;
- Transparência;
- Redução de tempo e custo;

- Registo de dados;
- Privacidade;
- *Smart Contracts*.

De todos os possíveis casos de uso nas diferentes indústrias, em que os benefícios listados anteriormente são enquadrados de acordo com o negócio, existem setores em que a utilização do *Blockchain* se ajusta de melhor forma e nas quais já existem vários casos de uso [32], entre os quais:

Tabela 4: *Blockchain* nos vários setores. Fonte: [32]

Setor	Descrição	Casos de Uso
Financeiro	Sistema financeiro descentralizado, digital, rápido, eficiente e principalmente seguro.	• *R3 CEV* • *Bank Hapoalim*
Saúde	Registo de dados médicos e históricos de pacientes de forma segura e, partilha desses dados com outros serviços.	• *Gem Health Network* • *Tierion*
Governamental	Sistema digital e distribuído que permite a redução de papel e corrupção, resultando assim numa estrutura legal e eficiente.	• *Delaware Blockchain Iniative* • *BitFury Group*
Imobiliário	Permite o armazenamento de todo o histórico de compra e venda de imobiliário sem necessidade de intermediários, permitindo assim total transparência, eficiência e redução de atividade fraudulenta.	• *Ubitquity*
Distribuição e Retalho	Registo do histórico do produto e transparência do mesmo, redução de tempo e processo mais eficiente, facilitando toda a atividade logística.	• *Provenance* • *Hijro* • *Skuchain*
Voto	Registo seguro da identificação e detalhes do voto, contagem automática dos votos e transparência nas estatísticas. Permitindo assim a redução de atividade fraudulenta no setor.	• *Follow My Vote*
Caridade	Transparência em todo o processo de doação em organizações de caridade.	• *BitGive Foundation*
Educação	Registo do histórico académico dos estudantes, permitindo segurança e transparência.	• *Sony Global Education e* IBM
Seguros	Permite às seguradoras obter dados credíveis em atividades de seguro, reduzindo tempo, custos e possíveis fraudes.	• *LenderBot*
IoT	Permite aos dispositivos comunicar através de um sistema distribuído de forma a automatizar a gestão de atualizações, bugs e gestão de energia.	• *Filament* • *IOTA* • *IoT Chain*
Armazenamento *cloud*	Sistema descentralizado para armazenamento de dados, assegurando a segurança de informação.	• *Storj* • *Sai* • *Swarm*

3.1.7 Tecnologias *Blockchain* relevantes

Existem inúmeras tecnologias para implementação de redes *Blockchain*, sejam elas públicas, privadas ou híbridas. Uma vez que a maioria destas tecnologias são *open-source* e se encontram disponíveis no *GitHub*, é possível ordenar as mesmas por popularidade, sendo o conceito de popularidade definido de acordo com o tamanho da comunidade. Assim, considerando que a comunidade participa ativamente no desenvolvimento e partilha de experiência da tecnologia, quando maior for a comunidade, mais robusta é a sua documentação, existirão menos *bugs* e o tempo para resolução dos mesmos é menor, existirão mais casos de uso e provas de conceito, entre outros.

Deste modo, para medir a popularidade do repositório de determinada tecnologia no *GitHub* foram utilizados os seguintes parâmetros:

- ***Watch*** – Número de desenvolvedores que pretendem receber notificações quando existe algum *pull request* e *issues* criados;
- ***Star*** – Número de indivíduos que está a seguir o repositório, mesmo não estando associado ao mesmo;
- ***Fork*** – Cópias criadas do repositório para desenvolvimento de funcionalidades ou correção de *bugs*.

Tabela 5: Comparação de popularidade entre as tecnologias *Blockchain*

Tecnologia	*Watch*	*Star*	*Fork*
Bitcoin [33]	3526	37881	22456
Ethereum [34]	2034	23075	8196
Hyperledger Fabric [35]	1108	8054	4591
Ripple [36]	472	3177	960
BigchainDB [37]	229	3152	621
Quorum [38]	334	3109	786
Corda [39]	274	2810	737
Stellar [40]	260	2062	626
MultiChain [41]	81	454	229
OpenChain [42]	82	433	173

De acordo com a Tabela 5 é possível comparar o nível de popularidade entre as 10 tecnologias *Blockchain* mais relevantes (dados de 12 de Abril de 2019). Para uma comparação mais aprofundada é possível descrever cada uma delas da seguinte forma:

- ***Bitcoin***:
    - O primeiro exemplo prático da tecnologia *Blockchain* e principal impulsionador da tecnologia;
    - Rede pública com criptomoeda associada (*Bitcoin*);
    - Com foco no setor financeiro;
    - *PoW* como mecanismo de consenso.

- ***Ethereum***:
    - Introdução à utilização de *smart contracts* em sistemas *Blockchain*;
    - Rede pública com criptomoeda associada (*Ether*);
    - Indicada para qualquer setor;
    - *PoW* como mecanismo de consenso;
    - *PoS* como mecanismo de consenso na versão 2.0, ainda numa fase inicial.
- ***Hyperledger Fabric***:
    - Tecnologia bastante versátil dado o seu nível de modularidade;
    - Permite a definição de permissões e dessa forma criar redes públicas, privadas ou híbridas sem qualquer criptomoeda associada;
    - Indicada para qualquer setor;
    - Mecanismo de consenso com *FT* (*Fault tolerance*).
- ***Ripple***:
    - Rede pública com criptomoeda associada (*XRP*);
    - Com foco no setor financeiro;
    - Mecanismo de consenso próprio (*Ripple protocol*).
- ***BigchainDB***:
    - Tecnologia com alto nível de escalabilidade, desenvolvida com foco em *Big Data*;
    - Permite a definição de permissões e dessa forma criar redes públicas, privadas ou híbridas, sem qualquer criptomoeda associada;
    - Indicada para qualquer setor relacionado com *Big Data*;
    - Mecanismo de consenso *BFT* (*Byzantine Fault Tolerant*).
- ***Quorum***:
    - Tecnologia baseada no *Ethereum* com características de uma rede privada;
    - Permite a definição de permissões e dessa forma criar redes públicas, privadas ou híbridas sem qualquer criptomoeda associada;
    - Indicada para qualquer setor;
    - Mecanismo de consenso *Raft-based* ou *Istanbul BFT*.
- ***Corda***:
    - Tecnologia que introduziu o conceito de "*pluggable notaries*" para melhorar a privacidade, escalabilidade e desempenho dos algoritmos;
    - Permite a definição de permissões e dessa forma criar redes públicas, privadas ou híbridas sem qualquer criptomoeda associada;
    - Indicada para o setor financeiro;
    - Mecanismo de consenso através de "*notaries nodes*".
- ***Stellar***:
    - Rede pública com criptomoeda associada (*Lumen*) (*XLM*);
    - Com foco no setor financeiro;
    - Mecanismo de consenso próprio *SCP* (*Stellar Consensus Protocol*).

- ***MultiChain***:
    - Permite a definição de permissões e dessa forma criar redes públicas, privadas ou híbridas sem qualquer criptomoeda associada;
    - Indicada para o setor financeiro;
    - Mecanismo de consenso próprio, idêntico ao *BFT*.
- ***OpenChain***:
    - Permite a definição de permissões e dessa forma criar redes públicas, privadas ou híbridas sem qualquer criptomoeda associada;
    - Indicada para qualquer setor;
    - Mecanismo de consenso "*Partionned Consensus*".

3.2 Bases de dados

As bases de dados são por norma selecionadas quando existe a necessidade de armazenar informação, sendo que nas bases de dados tradicionais são normalmente apresentadas as operações de *CRUD* (*create*, *read*, *update* e *delete*). No caso deste problema, apenas podem apresentar operações de escrita e leitura, uma vez que existe a necessidade de armazenar os dados de forma imutável. Além disso, são essenciais funcionalidades que garantam a segurança do sistema e dos dados armazenados, funcionalidades essas que possivelmente podem ser atingidas no caso da tecnologia funcionar de forma distribuída, uma vez que pode garantir a replicação de dados e mecanismos de tolerância a falhas.

De acordo com o apresentado, existem algumas tecnologias de bases de dados que permitem o armazenamento distribuído e a imutabilidade dos dados armazenados, garantindo assim a integridade dos dados, como é o caso das tecnologias *Apache CouchDB, Couchbase Server* e *Datomic Cloud*.

Estas tecnologias, detalhadas nas secções seguintes, podem-se comparar a nível de popularidade conforme o tamanho da sua comunidade, como já foi apresentado de forma idêntica na secção 24. Todavia, como nem todas as tecnologias apresentadas são *open-source* e se encontram disponíveis no *GitHub*, este conceito é definido conforme o número de questões associados a cada uma no *website* com a maior comunidade de desenvolvedores no mundo, o *StackOverflow*. Assim, de acordo com os dados recolhidos (dados de 10 de Junho de 2019) foi possível estruturar a seguinte tabela.

Tabela 6: Comparação de popularidade entre as tecnologias de bases de dados

Tecnologia	***Questões***
Apache CouchDB [43]	5628
Couchbase Server [44]	3069
Datomic Cloud [45]	361

3.2.1 *Apache CouchDB*

O *Apache CouchDB* é uma base de dados *open-source* não relacional orientada a documentos, no qual o armazenamento é realizado sobre uma estrutura de *b-tree*, e dada esta estrutura, não é possível editar ou eliminar dados. [46]

O *CouchDB* apresenta uma arquitetura replicada, idêntica às arquiteturas distribuídas, sendo assim estruturado sobre um *cluster* de nós. Esta estrutura garante assim mecanismos de tolerância a falhas no caso de algum nó da rede falhar. Utiliza mecanismos de *query* baseados em *JSON* para pesquisa de informação, e uma vez que garante um armazenamento imutável e não apresenta qualquer política de retenção, os dados são armazenados infinitamente. Além disto, não apresenta qualquer mecanismo de encriptação.

3.2.2 *Couchbase Server*

O *Couchbase Server* é uma base de dados *open-source* não relacional orientada a documentos e ao paradigma *key-value*, sendo que o armazenamento é realizado também sobre uma estrutura de *b-tree*, tal como o *CouchDB*. [47]

Apresenta uma arquitetura distribuída, mecanismos de tolerância a falhas e mecanismos de *query* através de *N1QL*. Possui uma política de retenção flexível que permite armazenar os dados por tempo definido, porém não possui qualquer mecanismo de encriptação.

3.2.3 *Datomic Cloud*

O *Datomic Cloud* é uma base de dados distribuída orientada a factos (*facts*), no qual os dados são armazenados de forma imutável nesses factos. É uma tecnologia paga disponível apena em ambiente *cloud* (*AWS*). [48]

Possui mecanismos de tolerância a falhas com o auxílio de serviços do *AWS* e apresenta mecanismo de query através da linguagem *Datalog*. Apresenta uma política de retenção e ainda mecanismos de encriptação através de chaves geridas pelo *AWS Key Management Service* (*KMS*).

3.3 *Apache Kafka*

Kafka é uma tecnologia em *Java* e *Scala*, *open-source* e desenvolvida pela *Apache*, foi construída inicialmente como um sistema de registo de *logs* e mais tarde tornou-se num sistema de mensagens sobre o conceito de *event sourcing*, contudo foi evoluindo de tal forma que atualmente possui utilizações distintas, intitulando-se como um sistema de *streaming* distribuído. [49] Assim, na sua versão atual, o *Apache Kafka* pode ser utilizado como sistema de mensagens, sistema de armazenamento ou sistema de processamento de *streams*.

Na verdade, o *Kafka* é muito mais do que um sistema de mensagens, uma vez que a sua arquitetura identifica-se mais com sistemas de ficheiros distribuídos ou base de dados do que propriamente com tecnologias como o *RabbitMQ* ou outros sistemas de mensagens, apresentando várias diferenças significativas [50]:

- O *Kafka* pode armazenar os dados eternamente ou por tempo definido, através da sua política de retenção, e esses dados podem ser lidos mais do que uma vez;
- O *Kafka* é construído sobre um sistema distribuído, é executado como um *cluster*, permitindo a expansão ou contração do mesmo e, replica os dados internamente, permitindo assim a tolerância a falhas e alta disponibilidade;
- O *Kafka* apresenta ótimo desempenho, estando preparado para o processamento de dados em tempo real de várias mensagens. Assim, é possível o tratamento de dados a um nível de abstração muito mais elevado.

Numa primeira abordagem, a tecnologia não possui qualquer semelhança com o *Blockchain*, contudo os conceitos base do *Kafka* são bastante idênticos aos do *Blockchain*, conceitos esses que apresentam valor para a solução a desenvolver.

"*Store streams of data safely in a distributed, replicated, fault-tolerant cluster.*" [49]

"*immutable sequence of records that is continually appended to—a structured commit log*" [49]

Tal como o *Blockchain*, *Kafka* possui o armazenamento de dados de forma distribuída, imutável e com mecanismos de tolerância a falhas (idêntico a alguns mecanismos de consenso em tecnologias *Blockchain*), garantindo assim a integridade dos dados. Ao contrário dos sistemas *Blockchain* possui um nível de escalabilidade e desempenho bastante elevado, porém não possui qualquer mecanismo de encriptação sobre os dados armazenados.

Apresenta uma comunidade bastante vasta, sendo que no seu projeto *open-source* disponível no *GitHub* [51], conta com 1013 desenvolvedores que pretendem ser notificados quando existe alguma alteração (*Watch*), 12365 desenvolvedores seguem o repositório (*Star*) e 6641 desenvolvedores que criaram uma cópia do projeto para desenvolvimento (*Fork*). Além disso, no *website StackOverflow* estão associadas 13994 questões à tecnologia. Todos os dados apresentados são referentes à data de escrita do documento (10 de Junho de 2019).

3.3.1 Arquitetura

A arquitetura do *Kafka* está organizada em tópicos, produtores, consumidores e *brokers*. Todas as mensagens são organizadas em tópicos, e desta forma enviadas e lidas para tópicos específicos. Assim, um produtor envia mensagens para determinado tópico e um consumidor lê mensagens desse mesmo tópico. Dada a estrutura distribuída do *Kafka*, o sistema é executado num *cluster*, no qual cada nó do *cluster* é chamado de *broker*. [52]

3.3.1.1 Tópicos

Um tópico está dividido em várias partições, permitindo assim replicar a informação de um tópico através de vários *brokers*, distribuindo assim a carga de processamento por vários nós e tornando desta forma o sistema mais eficiente. Cada mensagem dentro de uma partição encontra-se identificada por um *offset*, o que permite manter uma sequência imutável de mensagens ordenadas. A estrutura de um tópico pode ser representada através da Figura 9.

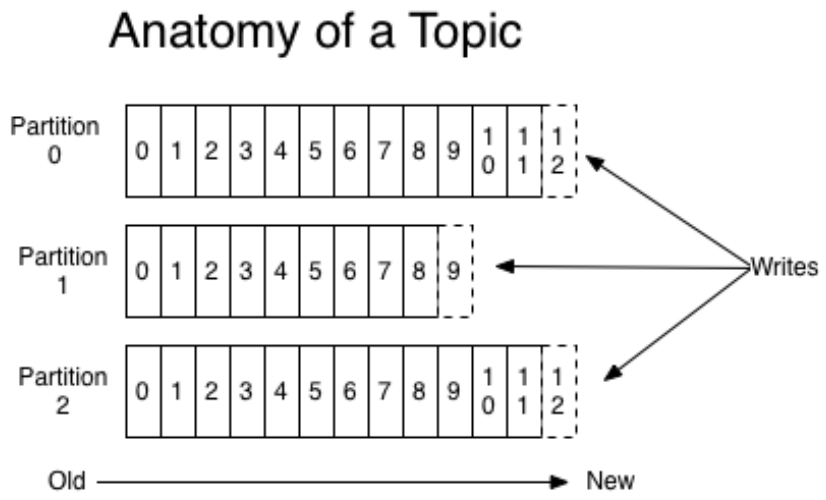


Figura 9: Estrutura de um tópico em *Kafka*. Fonte: [52]

Um tópico pode também ser analisado como um registo de *logs*, sendo possível eliminar cada registo/mensagem à medida que o mesmo é lido ou definir um tempo de retenção para todas as mensagens de um tópico.

Cada partição está ainda dividida por segmentos, sendo este o nível de abstração mais baixo da arquitetura do *Kafka*. Assim, cada segmento é um conjunto de mensagens de uma partição, o que permite armazenar as mensagens por segmentos, sendo cada segmento um ficheiro de armazenamento. Isto permite ao *Kafka* uma política de retenção mais flexível, uma vez que após o tempo de armazenamento definido ou de determinada capacidade de armazenamento atingida, é possível eliminar as mensagens por segmentos, o que é muito mais simples e eficiente do que eliminar várias mensagens de um único ficheiro caso a partição não estivesse dividida em segmentos.

3.3.1.2 *Kafka Brokers*

Cada *broker* do *cluster* mantém várias partições, que podem ser líderes ou seguidoras/réplicas (Na Figura 10 os líderes são representados a cinzento), sendo que o líder coordena a escrita e leitura de mensagens do tópico e redireciona a informação aos seguidores. Se uma partição líder falhar, outra partição seguidora assume o seu lugar. Este processo é realizado pelo *Zookeeper* através de um mecanismo de eleição baseado numa lista de *in-sync replicas* (ISR), assegurando assim tolerância a falhas e a disponibilidade do sistema.

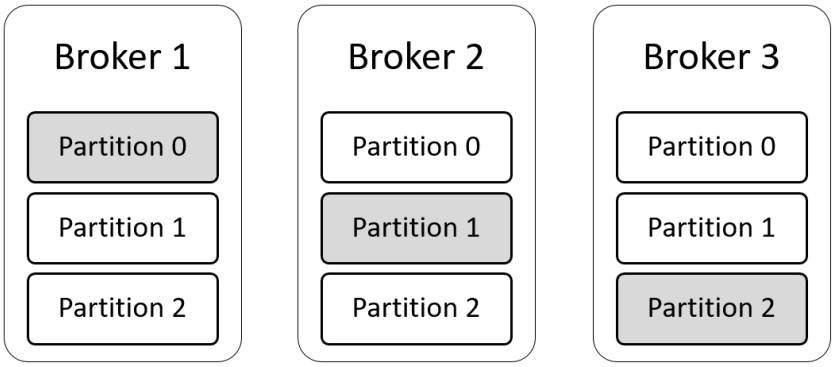


Figura 10: *Brokers* e partições no *Kafka*

Para um *cluster* de vários *brokers* também existe a designação de líder e seguidores, que tal como as partições líderes, são responsáveis por redirecionar a informação aos seguidores.

3.3.1.3 Produtores e Consumidores

Tal como foi referido, um produtor envia uma mensagem para determinado tópico, após isso o sistema deteta qual a partição líder que irá receber a mensagem e que por sua vez vai partilhar essa mesma mensagem com as partições seguidoras desse tópico.

O processo de leitura de mensagens, realizado pelo consumidor, vai ocorrer de forma diferente, sendo possível ler de qualquer partição. Uma vez que é possível ter vários consumidores a ler simultaneamente mensagens de um tópico, existe assim o conceito de grupo de consumidores, que permite definir para um conjunto de consumidores as partições que irão utilizar.

3.3.2 *Apache Zookeeper*

Dada a característica de sistema distribuído apresentada pelo *Kafka*, é necessário algum mecanismo que permita essa gestão de forma eficaz. Essa função é atribuída ao *Apache Zookeeper*, que através do protocolo *ZAB* (*Zookeeper Atomic Broadcast*) fornece ao *Kafka* mecanismos para gestão dos seus componentes e de tolerância a falhas, garantindo a segurança e disponibilidade do sistema e a integridade dos dados armazenados. [53]

Assim, o *Zookeeper*, utiliza o protocolo *ZAB* para assegurar que a replicação dos dados pelos vários *brokers* do *cluster* é executada corretamente, para isso recorre a um mecanismo de eleição do um nó líder, que é responsável por redirecionar informação, como já foi descrito na secção 3.3.1.2. Desta forma, através deste protocolo é possível assegurar a disponibilidade do sistema, e assim, no caso de um nó líder falhar, o mecanismo de eleição do nó líder é executado para a seleção de um novo líder e o sistema mantem-se em funcionamento. [54]

3.3.3 *Confluent Platform*

Uma vez que o *Kafka* é *open-source*, existem empresas que criaram plataformas mais completas sobre a tecnologia do *Kafka*, um exemplo disso é a *Confluent Platform*, uma plataforma que acrescenta outros componentes ao *Kafka*, como *Clients* de diferentes linguagens para interação com o *Kafka*, *Connectors* de diferentes bases de dados para ligação com o *Kafka*, *REST API* para interação com o *Kafka*, módulos de gestão dos vários nós do *Kafka* e *Zookeeper*, um mecanismo de *Schema Registry* para armazenamento de dados de forma estruturada, uma *engine SQL* (*KSQL*) com mecanismos de *query* sobre os dados armazenados no *Kafka*, entre outros, como é apresentado na Figura 11.

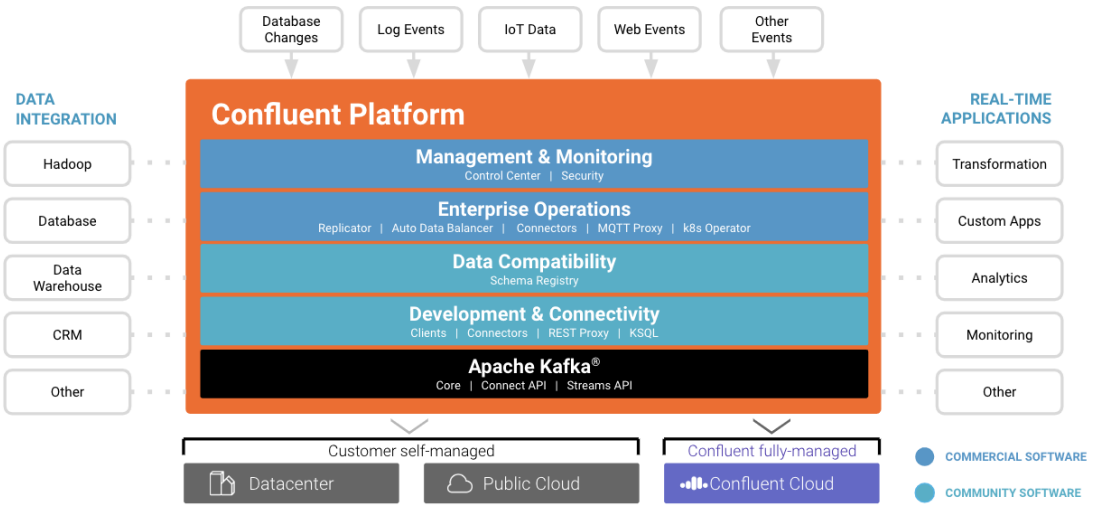


Figura 11: Componentes da *Confluent Platform*. Fonte: [55]

A *Confluent Platform* possuí uma versão grátis, na qual estão incluídos componentes bastante diferenciadores do *Kafka core* e, uma versão comercial e paga, com componentes para uma implementação a nível empresarial e total suporte por parte da empresa. Assim, a versão grátis desta plataforma apresenta algumas características interessantes, como é o caso dos componentes de *Schema Registry* e *KSQL*, que acrescentam características idênticas às das bases de dados tradicionais.

3.4 Encriptação

De modo a completar as tecnologias apresentadas anteriormente que não possuem encriptação, é necessário criar algum tipo de mecanismo que permita a encriptação dos dados inseridos no sistema, e da mesma forma a desencriptação dos mesmos quando são lidos, garantindo assim a confidencialidade da informação armazenada.

Assim, existiu a necessidade de analisar dois tipos de encriptação existentes, a encriptação simétrica e a encriptação assimétrica, para garantir a confidencialidade dos dados. A encriptação de "*one-way hash*" já foi explicada anteriormente (3.1.4.2) e, uma vez que se trata

de um tipo de encriptação no qual não é possível a desencriptação, não se aplica para este caso de uso.

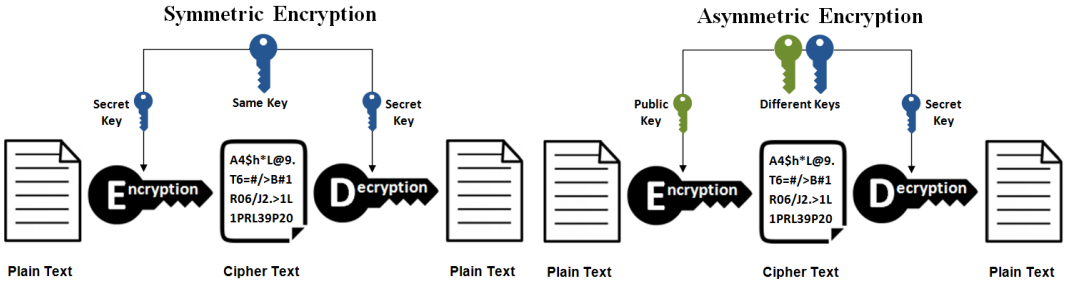


Figura 12: Encriptação simétrica e assimétrica. Fonte: [56]

De maneira a comparar os dois tipos de encriptação, representados na Figura 12, cada um deles foi detalhado de acordo com as características fundamentais da encriptação [56], como é apresentado na Tabela 7.

Tabela 7: Comparação da encriptação simétrica e assimétrica

Parâmetro	**Encriptação Simétrica**	**Encriptação Assimétrica**
Chaves	Uma chave para encriptação e desencriptação	Uma chave pública para encriptação e uma chave privada para desencriptação
Complexidade	Média	Alta
Desempenho	Alto	Médio
Segurança	Alto	Muito Alto
Algoritmos	*AES*, *RC4*, *DES*, *Blowfish*, *Twofish*	*RSA*, *Diffie-Hellman*, *El Gamal*, *ECC*, *DSA*

De qualquer modo, para qualquer tipo de encriptação é essencial utilizar pelo menos uma chave, chave essa que necessita de ser armazenada de forma segura para não comprometer a segurança da solução. Considerando que o serviço de encriptação é executado num servidor diferente do servidor no qual são armazenados os dados, o que já acrescenta um certo nível de segurança, existem várias formas de dar resposta a este problema:

- Armazenar a chave juntamente com o algoritmo de encriptação, sendo o processo mais simples mas pouco seguro;
- Armazenar a chave no mesmo servidor do que o algoritmo de encriptação, utilizando algum *key manager* local, como o *Java KeyStore*. Sendo um pouco mais complexo do que a primeira opção, mas mais seguro, uma vez que a chave não se encontra diretamente exposta no código;
- Armazenar a chave num formato encriptado numa base de dados externa ao algoritmo de encriptação, utilizando mecanismos para encriptação da mesma por máquina/utilizador, como é o caso do *Windows DPAPI*. Apresenta um nível de segurança superior à segunda opção, uma vez que a chave apenas pode ser encriptada e desencriptada em determinada máquina/utilizador, contudo, isso pode-se tornar numa restrição caso numa fase posterior seja necessário fazer uma migração de servidores;

- Armazenar a chave através de um *key manager* noutro servidor ou de serviços *key manager* na *Clould*, como é o caso do *Vault* e do *Amazon Key Management Service*. É o processo mais complexo e seguro porque é um mecanismo completamente independente do sistema, no entanto tem custos extra associados.

3.5 RGPD

O Regulamento Geral de Proteção de Dados (RGPD) [57], que entrou em vigor no dia 25 de Maio de 2018 e está presente na Carta dos Direitos Fundamentais da União Europeia, afirma que:

- Qualquer cidadão tem o direito de proteção da sua informação pessoal, de as aceder e alterar sempre que o solicitar;
- Qualquer um dos seus dados pessoais deve ser processado de forma justa para fins específicos e sempre com o seu consentimento;
- O cumprimento das regras presentes no regulamento deve estar sujeito a um controle, de forma a garantir que o sistema assegura a proteção dos dados.

Juntamente com o RGPD está presente o Direito ao esquecimento, o qual pode ser solicitado a qualquer momento e deve ser realizado na íntegra por parte do sistema. Este direito consiste em apagar completamente todos os dados pessoais de determinado indivíduo ou de vários indivíduos associados a uma entidade, sendo que todos os mecanismos para eliminação de dados devem ser explícitos.

Associado ao Direito do esquecimento e a partir do momento que o RGPD foi imposto, colocou-se a questão se o armazenamento imutável de informação e o RGPD podem coexistir, uma vez que o princípio de imutabilidade dos dados se opõe ao Direito ao esquecimento. Sobre esta questão estão associadas outras indecisões, dado que apenas é necessário aplicar o esquecimento sobre dados pessoais. Dessa forma, é essencial perceber que tipo de dados vão ser armazenados e se existe a necessidade de aplicar o RGPD.

Assim, de acordo com o RGPD [58], dados pessoais são todas as informações relativas a uma pessoa viva, identificada ou identificável, ou qualquer informação distinta que permita a identificação de determinada pessoa. Dados pessoais que tenham sido encriptados ou pseudonimizados, mas que possam ser utilizados para identificar uma pessoa, continuam a ser dados pessoais, no entanto, dados pessoais tornados anónimos deixam de ser considerados dados pessoais, porém têm de ser anonimizados de forma irreversível.

Posto isto, existem duas soluções possíveis para que sistemas de armazenamento imutável e o RGPD possam coexistir. A primeira é modelar o sistema para que possam ser armazenados dados pessoais e exista a funcionalidade de os eliminar de certa forma, e a segunda opção é tornar o sistema totalmente independente e sem necessidade de lidar com o RGPD.

Assim, com base em algumas soluções apresentadas após a introdução do RGPD [59], a Figura 13 representa uma solução para um sistema de armazenamento imutável (*Blockchain* utilizado como exemplo) que necessite de tratar dados pessoais e dessa forma aplicar o RGPD.

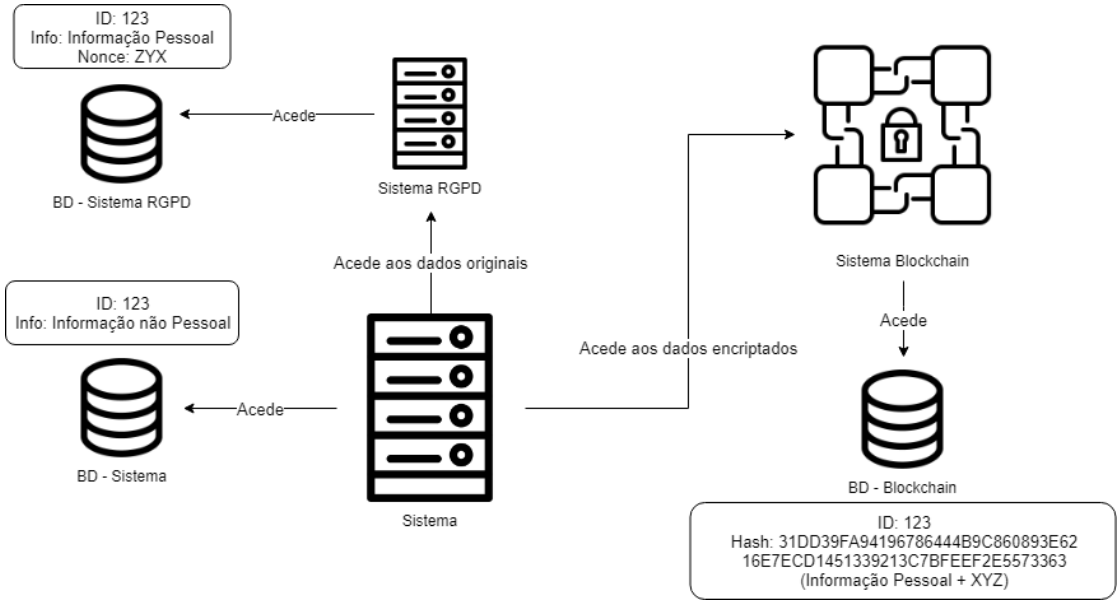


Figura 13: Diagrama conceptual de um Sistema *Blockchain* de acordo com o RGPD

De acordo com o diagrama, para o Sistema ter acesso aos dados pessoais e verificar a fiabilidade dos mesmos, teria de realizar um pedido ao sistema *Blockchain* conforme o identificador dos dados (ID), e iria obter a *hash* resultante da informação pessoal combinada com a *nonce*. Simultaneamente faria um pedido da idêntico ao Sistema RGPD para ter acesso aos dados originais, após obtidos os dados pessoais, é gerado um *hash* sobre esses dados combinados com a *nonce* e comparado com o *hash* retornado pelo Sistema *Blockchain*, verificando assim a integridade e fiabilidade desses dados.

A solução resolve a questão do Direito ao esquecimento, uma vez que, quando ativado basta apagar os dados do Sistema RGPD. Contudo, a utilização de uma *nonce* aleatória é essencial para garantir que a informação é anonimizada, caso contrário, o armazenamento de dados pessoais em formato de *hash* apenas garante que a informação é pseudonimizada e continua a ser abrangida pela legislação do RGPD. [60]

Por outro lado, acumulam-se outros problemas, uma vez que aumenta a complexidade arquitetural, a distribuição dos dados aumenta os pontos suscetíveis a ataques, é necessário definir quem administra o Sistema RGPD e a sua base de dados e, garantir a confidencialidade da chave utilizada na encriptação de informação.

A segunda solução seria tornar o sistema de armazenamento imutável totalmente independente da lógica do RGPD, armazenando apenas dados que por si só não permitam identificar uma pessoa. Desta forma, quando o Direito ao esquecimento for ativado, basta apagar os dados armazenados nos sistemas que utilizam o sistema de armazenamento imutável.

3.6 Soluções existentes

Lançado pela *Amazon* dia 28 de Novembro de 2018, o *Amazon QLDB* [61] é um serviço *cloud* da *Amazon AWS* que apresenta características idênticas às pretendidas no Sistema RDER. Apresentado como um *ledger database*, o serviço foi lançado inicialmente numa fase de *preview* e com acesso restrito para a comunidade até ao lançamento oficial, sendo representado pela Figura 14.

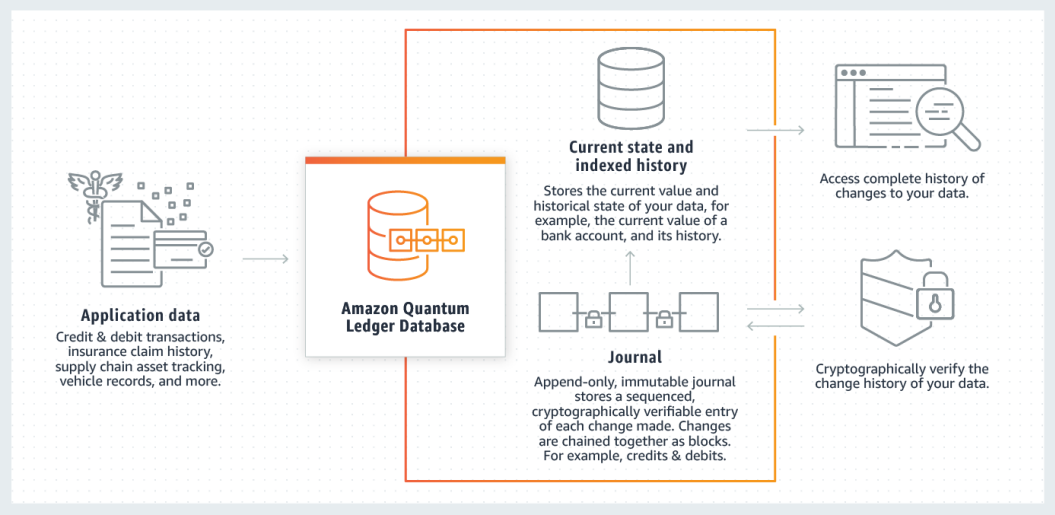


Figura 14: Funcionamento do *Amazon QLDB*. Fonte: [61]

Tal como é apresentado pela figura anterior, o *Amazon QLDB* possui as seguintes características:

- **Imutabilidade e transparência dos dados** – Os dados são armazenados num formato imutável, todo o histórico é registado e pode ser acedido a qualquer momento para verificar a integridade dos dados;
- **Encriptação dos dados** – Os dados são registados de forma segura utilizando o algoritmo de encriptação *SHA-256*, permitindo assim garantir a confiabilidade da informação;
- **Escalável** – Permite valores de escalabilidade duas ou três vezes superior a outras tecnologias *Blockchain*. Além disso é simples de implementar e toda a gestão é realizada por parte da *AWS*;
- **Suporte *SQL*** – Armazena os dados de forma estruturada através do *Amazon Ion* e disponibiliza uma *API* com suporte para *queries SQL*, facilitando assim o processamento dos dados.

4 Avaliação das tecnologias

Este capítulo tem como objetivo a análise das tecnologias pesquisadas e apresentadas no capítulo anterior. Numa fase inicial são apresentados os requisitos da solução e posteriormente é realizada uma análise detalhada às tecnologias de acordo com esses mesmos requisitos, por fim é apresentada a decisão relativa à encriptação de informação e a conclusão referente às possíveis soluções do problema.

4.1 Requisitos da solução

Para que as tecnologias apresentadas no capítulo anterior sejam avaliadas da melhor forma, sentiu-se a necessidade de definir os requisitos não funcionais da solução de modo a facilitar a escolha da tecnologia. Realizou-se uma divisão dos requisitos em requisitos base e requisitos específicos das tecnologias *Blockchain*, uma vez que as tecnologias *Blockchain* possuem características únicas relativamente às outras tecnologias apresentadas.

Assim, os requisitos base da solução, e aplicáveis a todas as tecnologias são os seguintes:

- Possuir características de armazenamento imutável;
- Possuir mecanismos de encriptação dos dados armazenados;
- Ser possível a sua utilização em vários casos de uso, para que seja possível criar um sistema abstrato e independente do modelo de negócio;
- Possuir um armazenamento seguro dos dados, por exemplo, através da sua replicação;
- Possuir mecanismos de acesso aos dados, através de linguagens de *query*;
- Possuir um mecanismo de tolerância a falhas / consenso seguro e de fácil implementação;
- Possuir um nível de escalabilidade e desempenho (principalmente de escrita) razoável, para que o sistema consiga suportar o aumento do volume de dados;
- Preferencialmente possuir compatibilidade com serviços de *cloud*, caso futuramente exista a necessidade implementar o sistema na *cloud*;
- Preferencialmente possuir uma política de retenção, para redução de recursos e custos;

- Ser grátis e preferencialmente *open-source*, para que não existam custos de utilização e seja possível realizar todas as alterações de *software* necessárias;
- Preferencialmente possuir uma comunidade de dimensão considerável para que exista suporte na correção de *bugs* e lançamentos de novas funcionalidades.

Os requisitos aplicados somente às tecnologias *Blockchain* são os seguintes:

- Permitir a definição de permissões nos nós participantes da rede, para que desta forma seja possível criar uma rede privada ou híbrida;
- Não existir nenhuma criptomoeda associada à tecnologia, para que não existam custos extra associados;
- Preferencialmente possuir o conceito de *smart contract*, o que permite a versatilidade do sistema.

4.2 Análise das tecnologias *Blockchain*

Conforme os requisitos apresentados na secção anterior, é possível limitar desde já a lista de tecnologias apresentada inicialmente. Assim, da lista inicial são descartadas as seguintes tecnologias:

- ***Bitcoin*** – Apesar de ser a tecnologia mais popular e a impulsionadora do conceito, e de possuir a maior comunidade entre todas, é bastante básica e limitada. Não cumpre alguns dos requisitos mais importantes, como o facto de não permitir a atribuição de permissões uma vez que se trata de uma rede pública, tem associada uma criptomoeda e utiliza o *PoW* como mecanismo de consenso, sendo este bastante dispendioso;
- ***Ethereum*** – Apesar de se tratar de uma tecnologia bastante robusta, com uma vasta comunidade e com uma aplicabilidade sobre vários casos de uso, peca pelo facto de não permitir a definição de permissões, estando assim limitada à sua rede pública. Tal como o *Bitcoin* possui o mecanismo de consenso *PoW*;
- ***Ripple*** **e** ***Stellar*** – Mesmo apresentando mecanismos de consenso seguros e diferentes do *PoW*, partilham algumas características com *Bitcoin* e *Ethereum*, tal como o facto de operarem apenas sobre uma rede pública e utilizarem uma criptomoeda;
- ***MultiChain*** **e** ***OpenChain*** – Ambas as tecnologias preenchem o requisito de possibilitar a definição de permissões, e de serem aplicáveis a vários setores. Porém, dada a sua reduzida comunidade não é totalmente seguro utilizar a mesma com a finalidade de produção, uma vez que a correção de bugs e de novas versões pode ser bastante lenta. Além disso, apesar do *MultiChain* ter suporte para os serviços *cloud* da *SAP*, tecnologia utilizada pela empresa, tem elevados custos associados à sua licença comercial, custos esses que não se encontram presentes em outras tecnologias.

Após definidos os requisitos e os mesmos serem utilizados como fatores limitadores na lista de tecnologias inicial, foi assim possível obter as tecnologias que se adequam aos requisitos

apresentados, dando assim resultado a uma lista de quatro tecnologias, sendo elas ***Hyperleder Fabric***, ***BigchainDB***, ***Quorum*** e ***Corda***.

Para que seja possível comparar as quatro tecnologias apresentadas anteriormente, realizou-se uma pesquisa detalhada sobre cada uma delas, pesquisa essa que se focou nos seguintes parâmetros, necessários para se perceber como cada tecnologia pode dar resposta aos requisitos apresentados:

- **Setor** – Identificação do setor para o qual a tecnologia foi concebida;
- **Linguagem** – Linguagem de programação na qual foi desenvolvida;
- ***Database State*** – Representa o tipo de dados armazenados no sistema;
- **Política de retenção** – Identificação de que a tecnologia permite reter os dados apenas por determinado período de tempo;
- **Arquitetura Modular** – Identificação de que a tecnologia permite uma arquitetura modular, permitindo assim a modelação do sistema para qualquer abordagem;
- ***Smart Contracts*** – Identificação de que a tecnologia possui o conceito de *Smart Contracts* por predefinição;
- **Algoritmo de Consenso** – Identificação do algoritmo de consenso utilizado;
- **Nível de Escalabilidade** – Representa o quão escalável pode ser o sistema, isto é, como é que o sistema reage à medida que o volume de dados a serem processados aumenta, neste caso a escrita e leitura de dados (*Throughput*). O nível de escalabilidade de cada tecnologia foi obtido através da pesquisa e análise de testes de desempenho sobre sistemas reais simulados (vários nós e mecanismo de consenso aplicado), e apesar de não serem diretamente comparáveis é possível classificar cada um deles através de uma escala mais genérica. Considerou-se assim quatro classificações a atribuir pelas quatros tecnologias em análise, Baixo, Médio, Alto e Muito Alto;
- **Plataformas *cloud*** – Platafomas *cloud* que têm suporte para a tecnologia, permitindo assim a fácil implementação da mesma em ambiente *cloud*. Apesar de ser possível a implementação de qualquer tecnologia desde que a plataforma *cloud* possua serviços de *Cloud Computing*;
- **Curva de Aprendizagem** – Representa a dificuldade de compreensão técnica da tecnologia relativamente ao tempo despendido, sendo que os resultados são baseados nas opiniões da comunidade, na opinião própria durante a pesquisa das tecnologias e, na quantidade de documentação e tutoriais disponíveis. Considerou-se assim as seguintes classificações para a curva de aprendizagem de modo que a tecnologia seja razoavelmente compreendida:
    - Tênue – Representa uma aprendizagem de curto período;
    - Intermédia – Representa uma aprendizagem de médio período;
    - Acentuada – Representa uma aprendizagem de longo período.

Considera-se que as quatro tecnologias apresentam todas as características fundamentais do *Blockchain*, como o armazenamento imutável e seguro dos dados, não possuem qualquer tipo de criptomoeda associada e que permitem a definição de permissões, sendo assim possível criar

redes privadas e híbridas. Assim, a Tabela 8 representa a pesquisa realizada para os parâmetros listados anteriormente.

Tabela 8: Comparação das tecnologias *Blockchain*

	HyperLedger Fabric [62]	***BigchainDB*** [63]	***Quorum*** [64]	***Corda*** [65]
Setor	Geral	Geral (*Big Data*)	Financeiro	Financeiro
Linguagem	*Go*	*Python*	*Go*	*Kotlin*
Database State	Dados do *Ledger*	Dados de transações	Dados da conta	Dados arbitrários
Política de retenção	Não	Não	Não	Não
Arquitetura Modular	Sim	Não	Não	Não
Smart Contracts	Sim	Não	Sim	Sim
Algoritmo de Consenso	*Kafka FT*	*BFT*	*Raft-based* ou *Istanbul BFT*	Verificação entre *notaries nodes*
Nível de Escalabilidade	Alto	Muito Alto	Médio	Baixo
Plataformas *cloud*	*AWS, Azure, SAP,* IBM, *Oracle* e *Alibaba*	Nenhuma	*Azure*	*Azure*
Curva de Aprendizagem	Acentuada	Intermédia	Intermédia	Intermédia

4.2.1 *HyperLedger Fabric*

Hyperledger Fabric é um *framework Blockchain* e parte do grupo de projetos da *Hyperledger*, mantido pela *Linux Fundation*. Trata-se de uma tecnologia que permite criar redes *Blockchain* para qualquer setor, existindo a possibilidade de aplicar permissões sobre os nós da rede.

Todos os dados do *ledger* são armazenados, existindo duas opções de armazenamento possíveis, o *LevelDB* e o *CouchDB*. O *LevelDB* é a tecnologia de armazenamento por predefinição e está integrada nos módulos *peer* de um nó da rede, o *CouchDB* é uma alternativa de fácil implementação e que permite o registo dos dados da *ledger* em formato *JSON* e com suporte para *queries* de maior complexidade.

Dada a sua estrutura modular, permite mais versatilidade no desenho da arquitetura do sistema e desta forma criar soluções completamente independentes do modelo de negócio ou modular a solução a um modelo de negócio específico, diferenciando-a assim de outras tecnologias no mercado.

Não possui qualquer política de retenção, mas apresenta o conceito de *Smart Contracts* através de uma abordagem própria, o *Chaincode*. O *Chaincode* é um programa que pode ser escrito em várias linguagens (*Go, node.js* ou *Java*) e é executado num *Docker container* isolado dos outros processos da rede.

Por predefinição, o sistema é implementado com o mecanismo de consenso *Solo*, como é apresentado pela tecnologia, existindo apenas um nó responsável pelo consenso na rede, contudo a documentação define que este mecanismo apenas deve ser usado para provas de

conceito uma vez que não garante qualquer validação dos dados inseridos na rede, não sendo assim apropriado para a fase de produção. Para uma fase de produção deve ser utilizado um mecanismo de consenso com *Fault tolerance* através do *Kafka*, que apesar de não ser tão seguro como o *BFT* (presente no *BigchainDB*), oferece segurança e robustez à rede.

De acordo a análise dos testes de desempenho [66] [67], a tecnologia apresenta valores distintos de *Throughput* conforme a base de dados em utilização (*LevelDB* ou *CouchDB*), sendo que com o uso de *CouchDB* o desempenho diminui, contudo apresenta-se bastante aceitável para ambos.

De entre todas as tecnologias é a que apresenta suporte oficial para um maior número de plataformas *cloud*, e dada a sua complexidade arquitetural é notavelmente a tecnologia com a curva de aprendizagem mais acentuada e que apresenta a maior dificuldade de implementação, apesar de disponibilizar uma documentação bastante completa.

A tecnologia permite a implementação de vários módulos adicionais, estando disponíveis vários componentes oficiais que facilitam a criação e manutenção da rede:

- *HyperLedger Composer* – Possibilita a criação de uma rede *Blockchain* com *JavaScript*, não sendo assim necessário configurar todos os parâmetros da rede através de *Hyperledger Fabric core*, como o modelo de dados, as permissões e a lógica de negócio. Permite também a geração de uma *REST API* de acesso à rede e ainda uma aplicação *web* em *Angular*;
- *HyperLedger Cello* – Utilizada para a manutenção da rede, permitindo a consulta dos vários nós da rede, a implementação de novos nós, etc;
- *HyperLedger Explorer* – Utilizada para a consulta dos dados da *ledger* na rede;
- *HyperLedger Caliper* – Ferramenta para realização de testes de desempenho na rede.

4.2.2 *BigchainDB*

BigchainDB foi criado com foco no alto nível de escalabilidade (*Big Data*), um dos maiores problemas quando se tratam de sistemas *Blockchain*, assim é possível desenvolver redes *Blockchain* que consigam manter o tempo de resposta do sistema minimamente estável mesmo com o aumento do volume de dados. A tecnologia permite também definir permissões para cada nó da rede.

Os dados das transações do *ledger* são armazenado em formato *JSON* numa base de dados *MongoDB*, em execução num dos módulos de cada nó da rede. Dado que se trata de uma tecnologia mais simples, não existe qualquer conceito de *Smart Contracts* nem apresenta qualquer política de retenção.

O mecanismo de consenso é realizado pelo *Tendermint*, uma tecnologia de consenso baseado no *BFT* que é executada num dos módulos de cada nó da rede. A utilização por predefinição do *Tendermint* no *BigchainDB* torna-o numa das tecnologias com o mecanismo de consenso mais seguro, com melhor desempenho e de fácil implementação.

Com base nos testes de desempenho apresentados pelos próprios desenvolvedores do *BigchainDB* [68], e como esperado, é a tecnologia que apresenta os melhore resultados de *Throughput* numa simulação de um ambiente real, mesmo com a complexidade do mecanismo de consenso presente.

Apesar de não suportar oficialmente nenhuma plataforma *cloud*, é possível implementar uma rede *Blockchain* em *BigchainDB* em qualquer plataforma *cloud* com suporte para *Cloud Computing*, como já foi referido anteriormente. Apresenta uma curva de aprendizagem intermédia, sendo bastaste acessível relativamente a outras tecnologias como o *Hyperledger Fabric*.

Existem diversos módulos oficiais e não oficiais para interação com uma rede construída com *BigchainDB*, além disso existem módulos não oficiais para consulta dos dados do *ledger* (*Dashboard*) e para realização de testes de desempenho (*Benchmark*).

4.2.3 *Quorum*

Tecnologia produzida pela empresa financeira *J.P.Morgan*, naturalmente com foco no setor financeiro, o que pode ser uma desvantagem neste caso de uso. Desenvolvida com base do *Ethereum*, mas apresentando algumas diferenças significativas, uma vez que é possível definir permissões na rede e não utiliza o mecanismo de consenso *PoW*.

Cada nó da rede possui dois tipos de armazenamento, um público para os dados partilhados na rede e outro privado com dados privados de cada nó. Tanto o armazenamento público como o privado possuem os dados de contas com as respetivas transações através do formato *key-value*.

Uma vez que a tecnologia foi desenvolvida sobre o *Ethereum* existe o conceito de *Smart Contracts* tal como no *Ethereum*, utilizando da mesma forma a linguagem de programação *Solidity*, própria para a escrita de *Smart Contracts*. Contudo, não possui qualquer política de retenção.

É possível implementar o mecanismo de consenso *Raft-based* ou *Istanbul BFT*, sendo que o *Raft-based* apresenta melhor desempenho mas não apresenta *BFT*, enquanto o *Istanbul BFT* apesar de não apresentar um desempenho ao nível do *Raft-based*, possui *BFT*.

Relativamente ao desempenho da tecnologia, e tal como referido anteriormente, varia conforme o mecanismo de consenso em utilização. Assim, com base na análise dos testes de desempenho para os dois protocolos de consenso [69] [70] [71], é possível classificar ambos como aceitáveis, contudo é notável que os testes com o mecanismo *Raft-based* apresentam melhores resultados.

A plataforma *cloud* do *Azure* possui suporte oficial para a tecnologia, apresenta uma curva de aprendizagem intermédia e possui vários módulos não oficiais que facilitam a criação e manutenção de uma rede:

- *Quorum Maker* – Ferramenta para implementação de novas redes, criação e adição de nós na rede, consulta de transações e dados da rede, etc;
- *Quorum Blockchain Explorer* – Ferramenta para consulta dos dados públicos e privados da rede;
- *Quorum-Genesis* – Facilita a criação do *genesis* file, simplificando assim a configuração da rede.

4.2.4 *Corda*

A tecnologia *Corda* foi apresentada em 2016, como uma tecnologia *Blockchain* para o setor financeiro com possibilidade de definir permissões na rede, tal como a tecnologia *Quorum*. Em 2018 apresentou a sua versão paga, o *Corda Enterprise*, com funcionalidades adicionais que permitissem a sua implementação em casos de uso de nível empresarial.

O armazenamento de dados ocorre sobre uma base de dados *H2*, onde são registados dados arbitrários, sendo desta forma possível a execução de *queries SQL* complexas. Possui o conceito de *Smart Contract* que se divide em três partes, a parte lógica representada pelo código a ser executado, o estado dos objetos representado pelos dados armazenados no *ledger* e por comandos extra que permitem dar versatilidade às transações. Não possui nenhuma política de retenção.

O mecanismo de consenso é realizado por *notaries nodes*, e como é apresentado na documentação é possível a integração de vários algoritmos de consenso conforme os requisitos do sistema, tal como *RAFT* ou *BFT*.

De acordo com a análise dos testes de desempenho da tecnologia [60], verificou-se que a tecnologia apresenta um valor razoável de *throughput*, porém os resultados analisados correspondem à versão paga, o *Corda Enterprise*. De acordo com os resultados de desempenho obtidos pela comunidade, a versão *open-source* da tecnologia apresenta um desempenho insatisfatório relativamente à versão paga, existindo assim um grave problema de escalabilidade.

Ao contrário das outras tecnologias em análise, o *Corda* não possui nenhuma ferramenta externa que facilite a criação e manutenção de uma rede *Blockchain*.

4.2.5 Síntese

Apesar das tecnologias apresentadas preencheram os requisitos propostos para o sistema RDER, o uso de tecnologias *Blockchain* foi reconsiderado, uma vez que para a solução idealizada, existiria uma rede com vários nós, porém apenas um desses nós introduziria dados na rede, tornando assim o sistema *Blockchain* num sistema centralizado sobre o único nó a inserir dados.

Foi possível chegar a esta conclusão após a análise minuciosa do funcionamento do *Blockchain* e das tecnologias baseadas no mesmo, dos requisitos propostos para a solução e da análise de

outras abordagens. Além disto foram utilizadas várias árvores de decisão, para se perceber se realmente seria sensato a utilização de *Blockchain* de acordo com os requisitos da solução. [73]

Um exemplo de árvore de decisão utilizada e da qual se concluiu que não existe a necessidade de utilizar *Blockchain*, foi através do modelo *B. Suichies*, apresentado na Figura 15. Além deste modelo, foram utilizados outros que apresentavam a mesma questão do acesso partilhado de escrita de dados, do qual a solução a desenvolver não necessita.

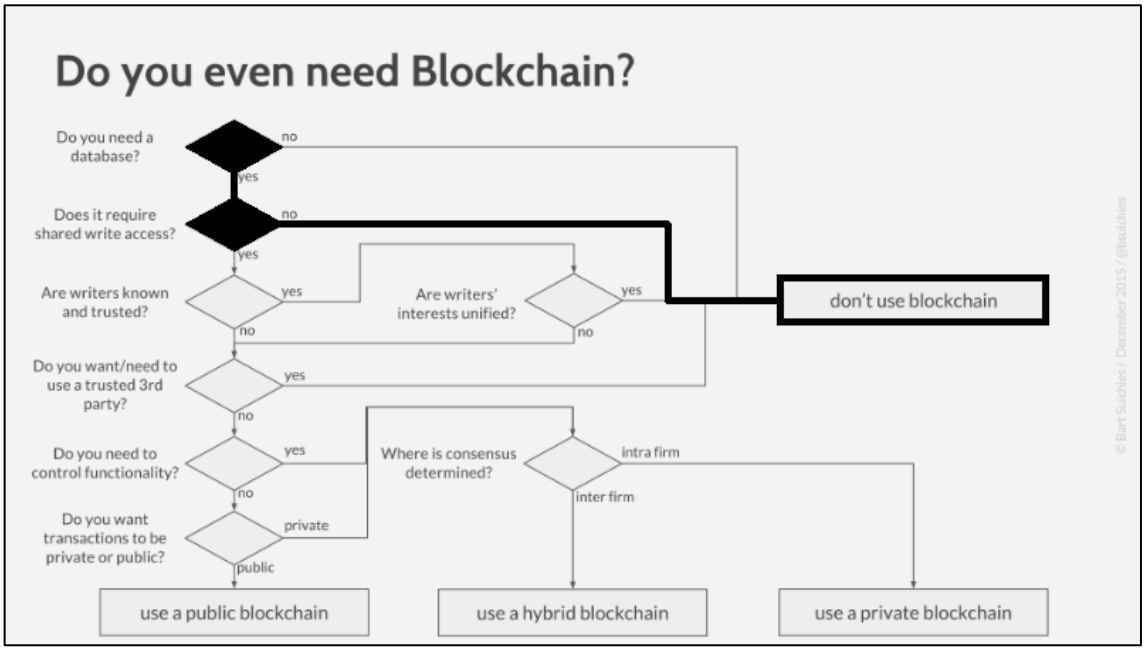


Figura 15: Modelo *B. Suichies*

Desta forma, ao desenvolver uma solução baseada em tecnologias *Blockchain*, não seria possível utilizar todo o potencial que as tecnologias *Blockchain* têm para oferecer. Uma vez que, ao centralizar o registro de dados num só nó, todos os mecanismos de consenso do qual estas tecnologias se baseiam não teriam qualquer utilidade, e desta forma seriam gastos recursos desnecessário num sistema que pode ser desenvolvido através de outras tecnologias.

4.3 Análise das tecnologias de bases de dados

As tecnologias de bases de dados apresentadas na secção 3.2 possuem características de armazenamento de dados de forma distribuída e imutável, enquadrando-se assim dentro dos requisitos esperados na solução. Contudo, diferem em vários aspetos e desse modo existiu a necessidade de as comparar conforme os requisitos enumerados na secção 4.1, como é apresentado na Tabela 9.

Tabela 9: Comparação das tecnologias de bases de dados

Requisito	***CouchBD***	***CouchBase***	***Datomic***
Armazenamento imutável	✓	✓	✓
Mecanismos de encriptação	✗	✗	✓
Versatilidade para vários casos de uso	✓	✓	✓
Armazenamento seguro	✓	✓	✓
Mecanismos de acesso aos dados	✓	✓	✓
Mecanismo de tolerância a falhas	✓	✓	✓
Nível de escalabilidade e desempenho razoável	✗	✓	✗
Compatibilidade com plataformas *cloud*	✓	✓	✓
Política de retenção	✗	✓	✓
Grátis e *open-source*	✓	✓	✗
Comunidade de dimensão considerável	✓	✓	✗

4.3.1 *Apache CouchDB*

De todas as tecnologias analisadas, o *CouchDB* é das que apresenta o maior número de pontos em falta. Apesar de possuir uma estrutura de dados imutável e de acrescentar segurança adicional a esses dados através da sua replicação pelo *cluster*, não possui qualquer mecanismo de encriptação da informação armazenada, sendo assim necessário implementar um mecanismo externo para essa tarefa.

Apresenta mecanismos de *query* baseado em *JSON* e mecanismos de tolerância a falhas, contudo apresenta resultados de desempenho muito pouco satisfatórios relativamente a outras tecnologias no mercado [74].

Possui compatibilidade com várias plataformas *cloud*, com é o caso do *Google Cloud*, *AWS*, *Azure* e *Oracle Cloud*, é grátis e *open-source*, possuí a comunidade de maior dimensão, no entanto não possui qualquer política de retenção.

4.3.2 *Couchbase Server*

Apresenta-se como a tecnologia mais completa entre as três, falhando apenas no mecanismo de encriptação. Apesar disso, possui uma estrutura de dados igual à apresentada pelo *CouchDB*, que como já foi referido, garante a imutabilidade dos dados armazenados, e garante também a replicação dos dados pelo sistema.

Possui mecanismos de *query* baseados na linguagem *N1QL* e mecanismos de tolerância a falhas. Além disso, apresenta resultados muito positivos em testes de desempenho, conseguindo atingir valores de *throughput* de cerca de 1 milhão de transações por segundo em cenários de inserção de dados. [75]

O *Couchbase* é grátis e *open-source*, possui uma comunidade com alguma dimensão, uma solução em ambiente *cloud* através de uma parceria com a *Google Cloud* e ainda uma política de retenção.

4.3.3 *Datomic Cloud*

O *Datomic* apresenta uma arquitetura distribuída e uma estrutura de dados orientada ao conceito de factos (*facts*), nos quais os dados são imutáveis e replicados pelo sistema.

Foi desenvolvido sobre a plataforma *cloud AWS* e dessa forma utiliza serviços da plataforma para garantir mecanismos de tolerância a falhas e a encriptação dos dados armazenados, sendo que as chaves são geridas pelo *AWS Key Management Service* (*KMS*).

Possui mecanismos de *query* através da linguagem *Datalog*, no entanto apresenta resultados de desempenho bastante insatisfatórios, uma vez que, excedendo 10 biliões de *datoms* (representação de um registo) por ano o nível de estabilidade do sistema diminui, o que representa cerca de 317 registos por segundo. [76]

Dada a sua implementação em ambiente *cloud*, a tecnologia tem custos associados e conta com uma comunidade de pequena dimensão, no entanto possui uma política de retenção.

4.3.4 Síntese

De acordo com as tecnologias apresentadas anteriormente não é imprudente considerar as tecnologias de bases de dados como soluções ao problema. Ambas as tecnologias apresentam armazenamento imutável, aplicabilidade a qualquer cado de uso, segurança no armazenamento de dados através da replicação dos mesmos, mecanismos de acesso aos dados, mecanismos de tolerância a falhas e compatibilidade com plataformas *cloud*.

Apresentam vários pontos em comum, no entanto o *CouchBase* diferencia-se por ser a tecnologia mais completa, apenas falhando no mecanismo de encriptação, que pode ser substituído por um serviço externo. O *CouchDB* mostrou ser uma tecnologia com muitas falhas de acordo com os requisitos apresentados, principalmente a nível de desempenho. Por fim, o *Datomic* também não apresentou bons resultados de desempenho, e apesar de ser a única tecnologia com um mecanismo de encriptação por predefinição, também é a única tecnologia paga e com uma comunidade muito limitada.

4.4 Análise do *Apache Kafka*

Como já foi referido, o *Kafka* apresenta várias características idênticas ao *Blockchain*, como a imutabilidade dos dados e o armazenamento dos mesmos de forma distribuída, todavia não possuir qualquer mecanismo de encriptação dos dados.

Apresenta uma versatilidade para vários casos de uso, permite um armazenamento seguro dada a replicação dos dados e mecanismos de acesso aos mesmos, apesar do acesso ser bastante limitado, o que pode ter um impacto negativo na utilização do sistema.

Possui o mecanismo de tolerância a falhas *ZAB* gerido pelo *Apache Zookeeper*, como já foi referido anteriormente. Além disso, apresenta valores de *throughput* bastante elevados, conseguindo suportar cerca de 2 milhões de inserções por segundo em testes realizados, o que garante excelente desempenho e permite um nível de escalabilidade elevado. [77]

Apresenta compatibilidade com várias plataformas *cloud* e possui uma comunidade de grande dimensão. Possui uma política de retenção flexível, que pode ser bastante útil para determinados casos de uso ou até para tratar de questões legais como o RGPD. É assim possível eliminar por completo um registo ou manter apenas o último valor (*value*) para determinado identificador (*key*) através do processo *log compaction*, visto que os dados são armazenados sobre uma arquitetura *key-value*. [49]

Possui ainda vários módulos oficiais e não oficiais para utilizações diversas, desde módulos de gestão do *cluster* do *Kafka* até módulos de ligações a bases de dados. Um dos módulos mais utilizados é o *Kafka Manager*, desenvolvido pela *Yahoo* e que permite uma gestão bastante completa do sistema. Outro módulo, o *Kafka Topics* UI, desenvolvido pela *Landoop*, permite a consulta dos dados armazenados no *Kafka* através de uma simples *dashboard*.

4.4.1 *Confluent Platform*

Como já foi apresentada anteriormente, a tecnologia *Confluent Platform*, apresenta várias características adicionais ao *Kafka core*, tornado assim a tecnologia ainda mais robusta e adaptável a mais casos de uso.

Uma característica simples mas de grande valor é a adição de uma *REST API* de acesso ao *Kafka*, não sendo necessário utilizar qualquer cliente externo para a ligação ao *Kafka*. Outros módulos mais recentes e que acrescentam características de bases de dados ao *Kafka* são o *Schema Registry* e o *KSQL*, tornando assim ainda mais viável a utilização do *Kafka* para armazenamento de dados.

4.4.2 Síntese

Assim, o *Apache Kafka* apresenta-se como uma solução bastante indicada para os requisitos definidos para a solução, apresentando apenas falhas nos mecanismos de encriptação e nos mecanismos de acesso aos dados, não possuindo na sua versão *core* qualquer processo que permita realizar *queries* complexas sobre dados armazenados, não sendo assim possível filtrar informação de forma eficaz.

Contudo, ambas as falhas podem ser contornadas, sendo que o mecanismo de encriptação pode ser implementado externamente tal como nas tecnologias de bases de dados que apresentam a mesma falha. Relativamente ao mecanismo de acesso aos dados, é possível solucionar esta falha através do uso da *Confluent Platform*, uma vez que a implementação do módulo *KSQL* permite a execução de *queries SQL* sobre os dados registados.

4.5 Encriptação

Conforme a pesquisa realizada aos tipos de encriptação existentes, e de forma a garantir a confidencialidade dos dados, é possível selecionar a encriptação simétrica como a mais adequada para a solução devido à sua velocidade de processamento em relação à encriptação assimétrica. Apesar da encriptação assimétrica apresentar um nível de segurança superior, o facto de ser um processo mais complexo e demorado pode comprometer o desempenho do sistema.

Para a comparação dos algoritmos de encriptação simétrica, foram utilizados os cinco finalistas do concurso para seleção do algoritmo *standard*, no qual o *Rijndael* foi vencedor e que, apesar de realizado em 2000, os algoritmos continuam a ser recomendados atualmente. Considerando que o nível de segurança dos finalistas é semelhante, o parâmetro de comparação foi a velocidade de processamento, de modo a não comprometer de qualquer forma o desempenho do sistema. Assim, os algoritmos analisados foram:

- ***Rijndael* (*AES*)** – Estruturado sobre uma rede de *SP* (*Substitution–permutation*), é executado em blocos de 128 bits e em rondas de 10, 12 e 14, para chaves de 128 bits, 192 bits e 256 bits, respetivamente;
- ***Twofish*** – Estruturado sobre uma rede de *Feistel*, é executado em blocos de 128 bits e em rondas de 16, para 128, 192 e 256 bits;
- ***Serpent*** - Estruturado sobre uma rede de SP, é executado em blocos de 128 bits e em rondas de 32, para 128, 192 e 256 bits;
- ***MARS*** - Estruturado sobre uma rede de *Feistel* do tipo 3, é executado em blocos de 128 bits e em rondas de 32, para 128, 192 e 256 bits;
- ***RC6*** - Estruturado sobre uma rede de *Feistel* do tipo 2, é executado em blocos de 128 bits e em rondas de 20, para 128, 192 e 256 bits.

Inicialmente foi necessário definir o tamanho da chave de encriptação, dado o seu impacto no desempenho dos algoritmos. Considerando então que atualmente o computador com maior poder de processamento consegue atingir os 143.5 Pentaflops [78] e o número total de combinações possíveis para uma chave de 128 bits do algoritmo *AES* é 3.4×10^{38}, são necessários cerca de 7.5×10^{16} anos, isto é, 75 quadriliões de anos, para que este mesmo computador consiga desencriptar através de força bruta qualquer mensagem encriptada.

Evidentemente que para uma chave maior, seja de 192 ou 256 bits, o nível de segurança aumenta consideravelmente, bem como o tempo necessário para desencriptação por força bruta. De qualquer forma, não existe a necessidade de tal nível de segurança visto que a chave de 128 bits apresenta segurança suficiente e é praticamente impossível de quebrar, além disso apresenta melhor desempenho relativamente a chaves de maiores dimensões.

De acordo com um estudo realizado aos algoritmos referidos anteriormente [79], foi notória a diferença de velocidade dos algoritmos *Rijndael* e *Twofish* em relação aos restantes, sendo os que apresentam a maior velocidade de processamento para chaves de 128, 192 e 156 bits. Contudo existe uma ligeira diferença entre os dois à medida que o tamanho da chave aumenta,

sendo que o *Twofish* se mantêm relativamente constante mas o *Rijndael* torna-se cerca de 20% mais lento para chaves de 192 bits e cerca 40% mais lento para chaves de 256 bits.

Mas, uma vez que não existe a necessidade de utilizar chaves com tamanho superior a 128 bits, o *Rijndael* apresenta-se como uma melhor opção relativamente ao *Twofish*, apresentando-se ligeiramente mais rápido, como é possível observar na Figura 16.

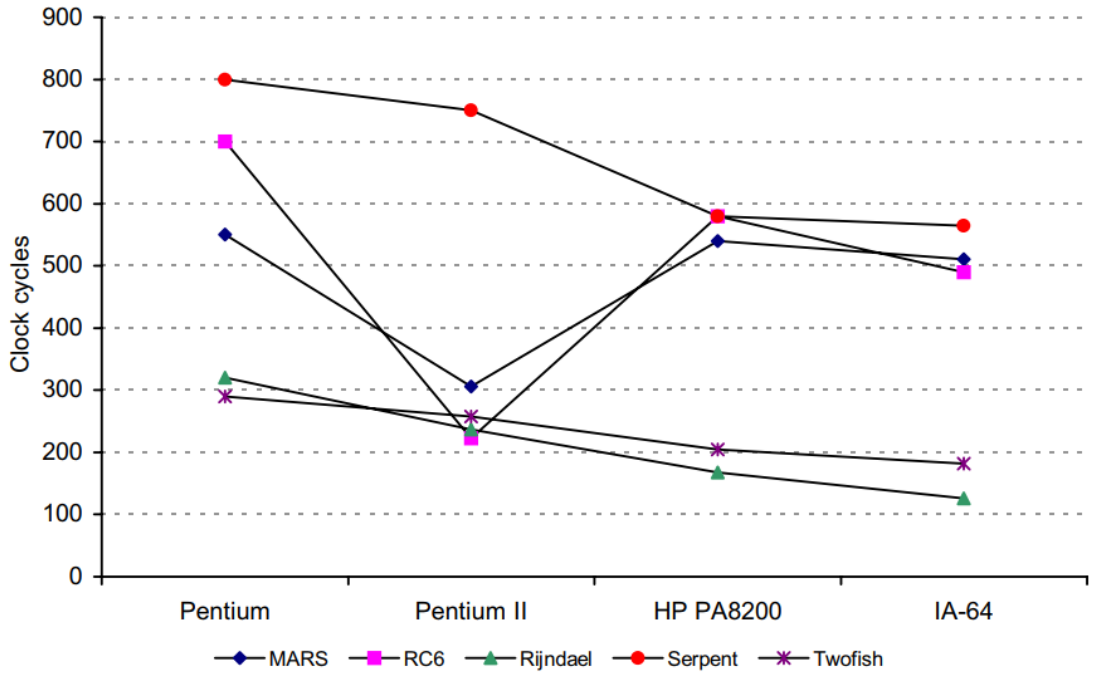


Figura 16: Comparação dos finalistas da *AES* para uma chave de 128 bits. Fonte: [79]

Além do *Rijndael* apresentar um desempenho ligeiramente superior ao *Twofish*, como foi selecionado como algoritmo de encriptação *standard*, tem vindo a ser otimizado desde então em vários *hardwares* e librarias de encriptação, tornando-o assim ainda mais robusto.

Foi ainda necessário escolher o modo de operação do algoritmo *AES*, dos quais foi considerado a sua velocidade de processamento com base nos valores disponibilizados pela biblioteca de encriptação *Crypto++* [80]:

- ***ECB*** – Modo predefinido para o algoritmo, simples e que permite paralelismo para maior velocidade, porém o menos seguro. Apresenta uma velocidade de 109 MiB/s para uma chave de 128 bits;
- ***CBC*** – É um dos modos mais comuns e considerado razoavelmente seguro. Permite apenas paralelismo na desencriptação. Apresenta uma velocidade de 109 MiB/s para uma chave de 128 bits, igual ao modo *ECB*;
- ***OFB*** – Permite que o código utilizado na encriptação também possa ser utilizado na desencriptação. Apresenta uma velocidade de 103 MiB/s para uma chave de 128 bits, o valor mais baixo de todos os modos;

- ***CFB*** – Idêntico ao *OFB* mas menos utilizado, apresentando uma velocidade de 108 MiB/s para uma chave de 128 bits;
- ***CTR*** – Um dos modos mais seguros e que permite paralelismo na encriptação e desencriptação, apresentando assim a maior velocidade para uma chave de 128 bits, no valor de 139 MiB/s.

Posto isto, e mais uma vez para não comprometer o desempenho do sistema, selecionou-se o modo *CTR* através do algoritmo *AES*, permitindo assim encriptar e desencriptar os dados de forma segura e sobretudo da forma mais rápida possível.

Relativamente aos mecanismos de armazenamentos das chaves de encriptação, a melhor opção é armazenar as mesmas num formato encriptado numa base de dados externa ao algoritmo de encriptação, encriptando-as por máquina, que apesar de não ser tão seguro como alguns serviços *cloud*, garante a segurança das chaves e não exige qualquer custo extra.

4.6 Conclusão

Após a análise detalhada das tecnologias *Blockchain* e de se debater a necessidade do seu uso, concluiu-se que não a sua utilização não é a mais apropriada para a solução a desenvolver, sendo que a decisão recai sobre a tecnologia de base de dados mais completa, o *Couchbase*, e o *Kafka*.

Assim, o *Couchbase* relativamente ao *Kakfa* apresenta mecanismos de acesso aos dados mais completos, mesmo comparado com a integração do *KSQL* no *Kafka*. Por outro lado, o *Kafka* apresenta um mecanismo de tolerância a falhas bastante robusto, uma comunidade de maior dimensão e de acordo com os resultados de desempenho de ambas [75] [77], para testes idênticos, o *Kafka* apresenta cerca do dobro do desempenho relativamente ao *Couchbase*, sendo a característica de maior relevância quando comparadas.

Para dar resposta à única característica não presente pelas tecnologias, foi necessário desenvolver um mecanismo de encriptação para que os dados sejam armazenados num formato ilegível, o que adiciona uma camada extra de segurança ao sistema RDER e o torna totalmente apropriado aos requisitos apresentados. Assim, o mecanismo que garantirá a confidencialidade dos dados utilizará o algoritmo de encriptação *AES* no modo *CTR*, sendo que as chaves utilizadas são armazenadas numa base de dados externa ao algoritmo de encriptação, encriptando as mesmas por máquina.

Deste modo, a utilização do *Apache Kafka* juntamente com o mecanismo de encriptação a desenvolver garante que todos os requisitos são preenchidos e torna a tecnologia mais indicada para solucionar o problema. No entanto, é viável utilizar o *Apache Kafka* através de várias abordagens, sendo assim possível construir o sistema RDER de três formas distintas:

- ***Kafka core*** – Sistema simples utilizando *Kafka* na sua versão *core*, com mecanismos básicos de acesso aos dados e uma política de retenção flexível;

- ***Kafka core* integrado com *Couchbase*** – Sistema complexo e que utiliza todo o potencial de armazenamento do *Couchbase*, permitindo assim o armazenamento estruturado de dados e *queries* mais complexas. A tecnologia do *Kafka* neste sistema permitiria a disponibilidade e escalabilidade do mesmo. Um exemplo de como este sistema se compara com os sistemas *Blockchain* é a tecnologia *Hyperledger Fabric*, que apresenta uma arquitetura idêntica, utilizando *Kafka* para o mecanismo de consenso, alta disponibilidade e escalabilidade e *CouchBD* para armazenamento de dados;
- ***Confluent Platform*** – Sistema mais complexo do que *Kafka core*, porém não tão complexo como *Kafka core* integrado com Base de dados. Através desta plataforma é possível usar todo o potencial do *Kafka core* e ainda de componentes extras desenvolvidos pela plataforma, dois dos quais acrescentam características de base de dados ao armazenamento do *Kafka*, como já foi referido anteriormente.

5 *Design* da Solução

No presente capítulo é realizada inicialmente a avaliação das soluções expostas no final do capítulo anterior, no qual é eleita a solução mais indicada. De seguida é apresentada a arquitetura da solução, sendo descrito em detalhe cada componente desenvolvido, desde o seu funcionamento interno até à sua interação com os restantes componentes, e por fim são apresentadas as arquiteturas alternativas.

5.1 Avaliação das soluções

Para uma comparação mais detalhada entre as soluções apresentadas na secção 4.6, foi construída a Tabela 10 na qual é possível comparar características essenciais para a solução, entre as quais:

- **Complexidade** – Complexidade associada ao desenvolvimento, implementação e manutenção da solução;
- **Desempenho** – Desempenho e nível de escalabilidade apresentado pela solução, afetado diretamente pela complexidade da mesma;
- **Custos** – Custos associados ao desenvolvimento, implementação e manutenção da solução, seja em servidores locais ou através de serviços *cloud*;
- **Acessibilidade a módulos extra** – Nível de acessibilidade a módulos extra, que apesar de qualquer solução permitir a integração de novos componentes, existem soluções às quais o acesso e integração de módulos é facilitado;
- **Política de retenção** – Possibilidade de definir o tempo de retenção dos dados;
- **Mecanismos de *query*** – Permite mecanismos de processamento de dados, por exemplo, via *queries SQL*;
- **Acesso via *REST API*** – Disponibiliza uma *REST API* como ponto de acesso ao sistema.

Tabela 10: Comparação das várias arquiteturas

Característica	***Confluent Platform***	***Kafka core***	***Kafka core + Couchbase***
Complexidade	Média	Baixa	Alta
Desempenho	Alto	Muito Alto	Médio/Alto
Custos	Médio	Médio/Baixo	Alto
Acessibilidade a módulos extra	Alta	Média	Média
Política de retenção	✓	✓	✓
Armazenamento estruturado	✓	✗	✓
Mecanismos de *query*	✓	✗	✓
Acesso via *REST API*	✓	✗	✗

Em suma, a utilização da arquitetura baseada na *Confluent Platfom*, permite que o nível de complexidade do sistema não seja exageradamente elevado, o que desta forma facilita o desenvolvimento e manutenção futura e garante assim um desempenho e escalabilidade do sistema razoável. Apesar dos custos dependerem da forma como o sistema é implementado e do tipo de *hardware* que o suporta, para o mesmo nível de desempenho, a solução arquitetural em questão apresenta-se como a mais equilibrada.

Uma vez que a *Confluent Platform* apresenta vários módulos adicionais ao *Kakfa*, existe facilidade na integração desses módulos relativamente às arquiteturas que não utilizam a plataforma, apesar da integração ser possível em qualquer arquitetura.

Por fim, apresenta simultaneamente características como a política de retenção original do *Kafka*, armazenamento estruturado de dados, mecanismos de *query* através da *engine SQL* apresentada pelo *KSQL* e a interação com o *Kakfa* através de uma *REST API*, sendo assim a solução mais indicada para os requisitos definidos.

5.2 Arquitetura

Com base na comparação realizada entre as três soluções arquiteturais para o Sistema RDER, concluiu-se que arquitetura estruturada sobre o *Kafka Confluent Platform* (versão 5.1.0) se apresenta como a mais indicada. Assim, com base nessa solução e como é apresentado na Figura 17, a arquitetura é composta por nove componentes, sendo que os quatro primeiros representam o sistema RDER:

- ***Kafka Node*** - Representa um nó a operar através do *Apache Kafka*;
- ***Confluent System*** - Representa o sistema que executa os componente do *Confluent Platform*, operando sobre o *cluster Kafka*;
- ***Access System*** - Responsável por realizar a ligação entre as aplicações que utilizam o sistema e os nós do *Kafka*;
- ***Access System DB*** – Base de dados do *Access System*;
- ***Web App*** – Representa a aplicação *web* para gestão do Sistema RDER e do Sistema *IoT*;
- ***Web App DB*** – Base de dados da *Web App*;

- ***IoT System*** - Representa o sistema *IoT*, utiliza o Sistema RDER e fornece dados para a *Web App*;
- ***Client App*** - Representa a aplicação que utiliza o Sistema RDER;
- ***Client Browser*** - Representa o *browser* que utiliza a *Web App*.

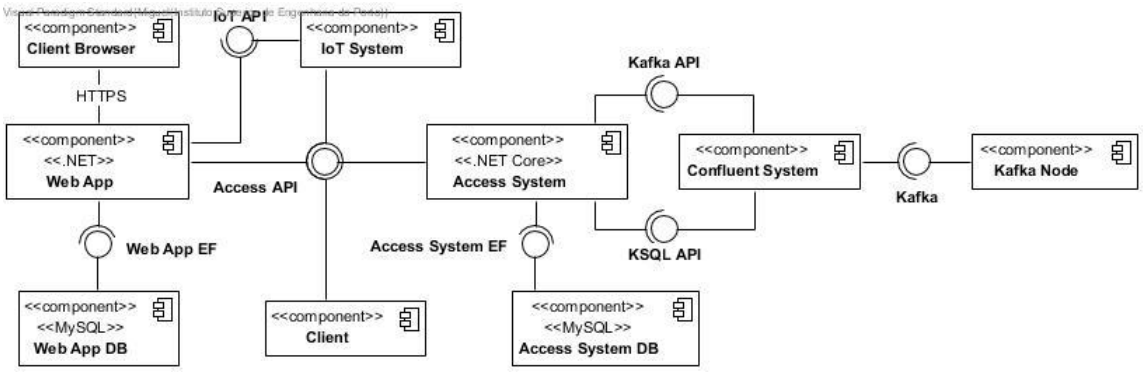


Figura 17: Diagrama de componentes de alto nível da solução

Na Figura 18 é possível representar com mais detalhe cada um dos componentes enumerados anteriormente.

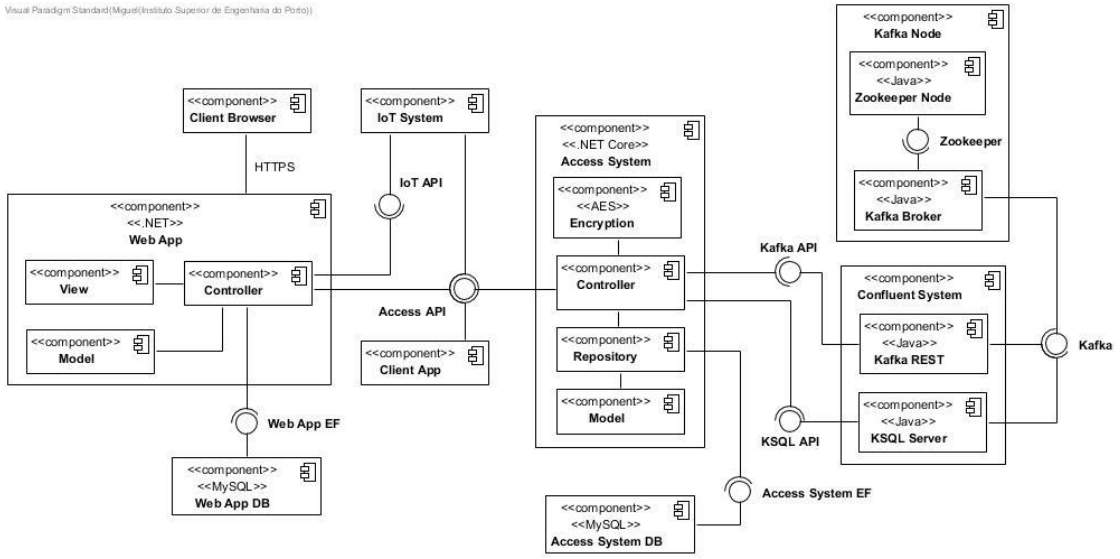


Figura 18: Diagrama de componentes de baixo nível da solução

A implementação da solução foi idealizada através de oito servidores com sistemas operativos *Linux* e *Windows*:

- ***Kafka Server*** - Três servidores com *Ubuntu Server* 18.04.2 *LTS*, cada um para cada nó do *cluster* do *Kafka*, através da execução dos componentes *core* do *Kafka*;
- ***Confluent System Server*** – Um servidor com *Ubuntu Server* 18.04.2 *LTS*, para execução dos componentes da *Confluent Platform*, *Kafka REST* e *Kafka Server;*
- ***Access System Server*** – Um servidor com *Windows Server*, para permitir o acesso ao Sistema RDER através da execução do componente *Access System*;

- ***Access System DB Server*** – Um servidor com *Windows Server*, para garantir o armazenamento dos dados do *Access System* através da execução do componente *Access System DB*;
- ***Web App Server*** – Um servidor com *Windows Server*, para permitir a gestão do Sistema RDER através da execução do componente *Web App*;
- ***Web App DB Server*** – Um servidor com *Windows Server*, para armazenar os dados da *Web App* através da execução do componente *Web App DB*.

Os servidores comunicam entre si através do protocolo *HTTPS* (*Hypertext Transfer Protocol Secure*), sendo que a informação é encriptada através do protocolo *SSL* (*Secure Sockets Layer*) para manter o nível de segurança do sistema. Os componentes desenvolvido em *.NET* são executados sobre um servidor IIS e as bases de dados representadas pelos componentes *Access System DB* e *Web App DB* são executadas em *SQL Server*. Definiu-se três nós no *cluster Kafka* e um fator de replicação de três, construindo assim a arquitetura mínima para garantir alta disponibilidade e consistência do Sistema RDER. [81]

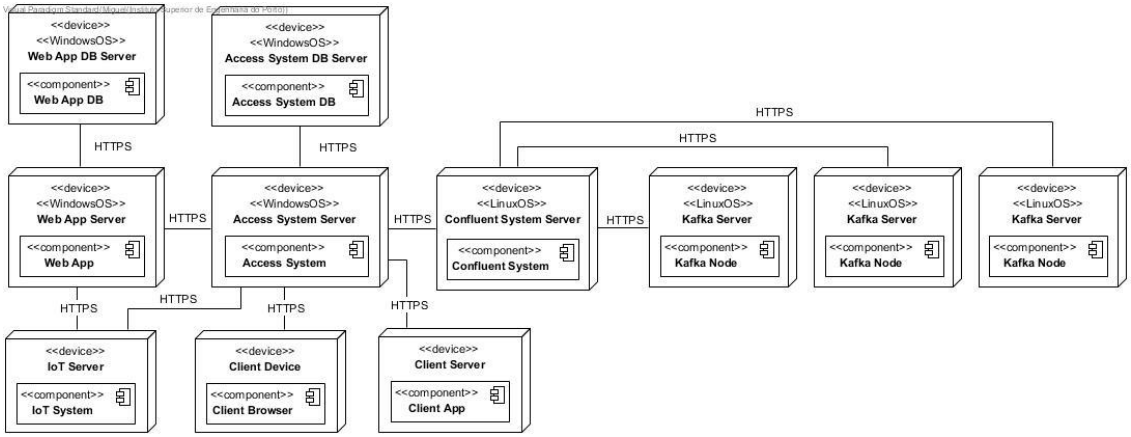


Figura 19: Diagrama de implementação da solução

Dos nove componentes apresentados anteriormente, que se complementam de forma a criar a solução final, apenas seis foram desenvolvidas no âmbito desta tese, sendo descritos nas próximas secções. Os restantes componentes foram criados por outros desenvolvedores ou já existiam.

5.2.1 *Kafka Node* e *Confluent System*

O componente *Kafka Node* é responsável por executar um nó *Kafka* (*Kafka Broker*) para o funcionamento do *cluster do Kafka*, e um *Zookeeper Node* para a gestão desse mesmo *cluster*.

No caso de ser necessário adicionar um novo nó (*Kafka Broker*) ao *cluster* do *Kafka*, basta executar os componentes do *Kafka Node* num novo servidor configurados conforme o *cluster*. Contudo, as partições já existentes no sistema não são automaticamente distribuídas por este novo nó, apenas as partições de tópicos que sejam criados posteriormente, apesar de ser possível faze-lo manualmente.

O *Confluent System* executa os componentes da *Confluent Platform* que atuam sobre o *cluster* de *Kafka Brokers*, o componente *Kafka REST* que permite o acesso ao sistema através de uma *REST API* e, o componente *KSQL Server* que permite o funcionamento da *engine SQL* desenvolvida para o *Kafka* e que também disponibiliza uma *REST API*. Ambas as *API's* disponibilizam inúmeros *endpoints* de acesso, no entanto apenas é necessário utilizar dois *endpoints* de cada uma, para acesso ao estado do *cluster* do *Kafka*, para registo de mensagens e simultaneamente para a criação de tópico, para a criação de *streams* e listagem de mensagens.

5.2.1.1 Modelo de dados

A estrutura das mensagens deve ser o mais abstrata possível relativamente ao modelo de negócio, sendo assim, apenas é necessário um identificador do registo, a data e hora do registo, o conteúdo desse registo e um identificador de quem o registou, podendo ser necessário para qualquer auditoria. Desta forma, as mensagens armazenadas podem ser acedidas de forma estruturada através do formato apresentado na Tabela 11.

Tabela 11: Estrutura de dados mapeada pela *stream*

ROWTIME	***ROWKEY***	***value***	***user***
1540254230041	1	2mRvZo4pUcb8BP/123LJEQ==	32452
1540254230081	2	AbjOwklv742e5N5Yqmp0ww==	54647
1540254230091	3	E34p41ifEKl+Z02/fMLnaQ==	93244
…	…	…	…

É possível estruturar a mensagem desta forma através da utilização do módulo *KSQL*, que permite a criação de *streams* e *tables* (difere das *streams* por armazenar apenas o último valor de determinada *key*) como camada de abstração às mensagens armazenadas nos tópicos, sendo que se utilizou apenas *streams*. Assim, o *KSQL* permite definir para cada *stream* uma estrutura dos dados em *JSON*, sendo que neste caso a estrutura é definidas pelo *value* e *user*, não existindo necessidade de incluir o *ROWTIME* e *ROWKEY* uma vez que são atributos predefinidos.

A Tabela 12 representa a estrutura dos dados armazenados num tópico para o conjunto de dados apresentados na tabela anterior, no qual o conteúdo na mensagem está formatado em *JSON* para possibilitar a seu mapeamento no módulo *KSQL*.

Tabela 12: Estrutura de dados do tópico

timestamp	***key***	***value***
1540254230041	1	{“value”: “2mRvZo4pUcb8BP/123LJEQ==”, “user”: 32452}
1540254230081	2	{“value”: “AbjOwklv742e5N5Yqmp0ww==”, “user”: 54647}
1540254230091	3	{“value”: “E34p41ifEKl+Z02/fMLnaQ==”, “user”: 93244}
…	…	…

Em suma, a estrutura foi definida apenas com o parâmetro *value* (informação encriptada de cada mensagem) e *user* (utilizador que registou a mensagem), sendo que os parâmetros

ROWTIME e *ROWKEY* são originalmente criados e associados respetivamente ao *timestamp* (precisão em milissegundos) e à *key* da mensagem. Desta forma é possível a execução de *queries* sobre as mensagens, permitindo a filtragem das mesmas por *key*, intervalos de tempo e utilizadores.

5.2.2 *Access System* e *Access System DB*

O acesso às operações do sistema é assim da responsabilidade do *Access System*, sendo realizado através de uma *REST API* disponibilizada pelo mesmo, o que permite a utilização por parte de qualquer aplicação que consiga executar pedidos *HTTP*. Todo o componente *Access System* foi desenvolvido em *.NET Core* (C#) e integrado com uma base de dados *MySQL* (*Access System DB*) para armazenamento dos dados de todo modelo de negócio, uma vez que são tecnologias utilizadas maioritariamente pela empresa e, deste modo, facilita o desenvolvimento e manutenção em situações futuras. A ligação ao *Kafka* é realizada através da *REST API* da *Confluent Platform* (*Kafka API*) para criação de tópicos e registo de mensagens, no entanto a criação de *streams* e o acesso aos dados armazenados é da responsabilidade da *REST API* do *KSQL* (*KSQL API*).

A *REST API* de acesso ao sistema opera em conjunto com um componente de encriptação e desencriptação de dados (*Encryption*), através do algoritmo *AES* (modo *CTR*), no qual é utilizado o *Windows DPAPI* para encriptação e desencriptação das chaves de 128 bits por máquina.

Utilizou-se uma arquitetura idêntica à apresentada pelo *MVC* mas sem o componente *View*, uma vez que não existe qualquer necessidade de interface gráfica. Utilizou-se também o padrão *Repository* criando assim uma camada de abstração entre a camada de negócio (*Business Logic Layer*) e a camada de acesso aos dados (*Data Access Layer*). O mapeamento das entidades armazenadas na base de dados é da responsabilidade do *Entity Framework Core*.

5.2.2.1 Modelo de dados

O modelo de dados do *Access System*, no qual as entidades são armazenadas na *Access System DB* é representado na Figura 20 pelas seguintes entidades:

- ***User*** – Identifica um utilizador que pode ser proprietário de uma ou mais instalações (*Sites*). E. g. Administrador do sistema;
- ***Site*** – Identifica uma instalação que pode ter um ou mais proprietários (*Users*) e um ou mais tópicos (*Topics*), além disso está associada ao utilizador (*User*) que a criou. E. g. Armazém 3 da empresa Teste;
- ***Topic*** – Representa um tópico do *Kafka*, que está associado diretamente ao utilizador (*User*) que o criou e a uma instalação (*Site*). E. g. Registo de dados críticos;
- ***AesKey*** - Representa uma chave de *AES* de 128 bits encriptada pelo *Windows DPAPI*, que é responsável pelo mecanismo de encriptação das mensagens registadas em todos os tópicos (*Topics*) de determinada instalação (*Site*).

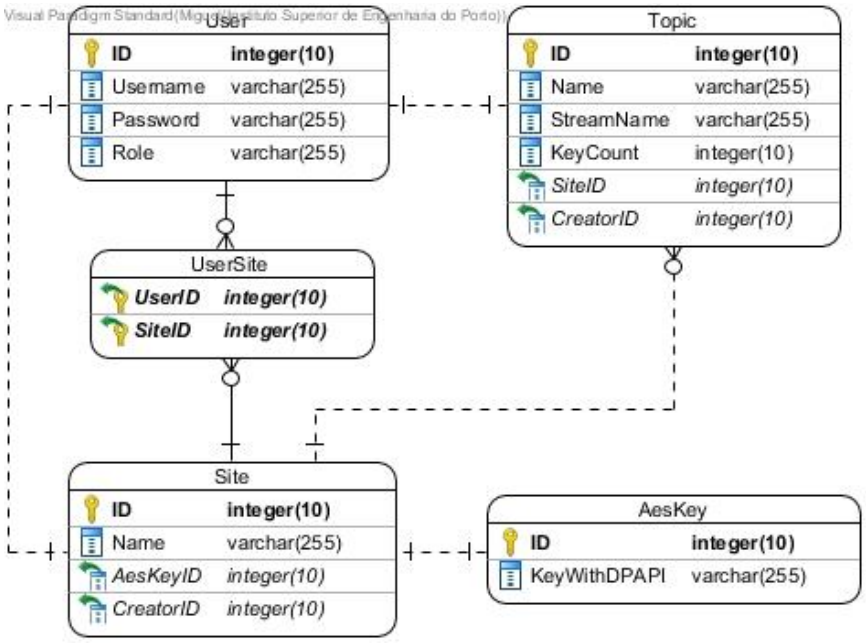


Figura 20: Modelo de dados do *Access System*

Associado à entidade *User* está ainda a propriedade *Role*, que representa a função desse utilizador no sistema, garantindo assim mecanismos de autorização. Deste modo, a autorização às várias operações do sistema foram definidas da seguinte forma:

- ***SuperAdmin*** – Representa o administrador do sistema. Possui acesso a todas as operações;
- ***CompanyAdmin*** – Representa o responsável por uma empresa cliente. Tem permissões para alterar os seus dados de acesso e para todas as operações relativas a instalações que seja proprietário, incluindo a criação de novas instalações;
- ***CompanyUser*** – Representa um funcionário da empresa cliente. Tem permissões para alterar os seus dados de acesso, permissões para listar instalações das quais seja proprietário e todas as operações relativas aos tópicos que pertençam a instalações das quais seja proprietário.

5.2.2.2 *REST API endpoints*

A *REST API* a ser desenvolvida para o componente *Access System* tem de integrar todos os métodos necessários para uma utilização completa do sistema por parte das aplicações que o integram, sejam serviços da empresa ou a *Web App* desenvolvida.

Não existe qualquer método para eliminar tópicos, uma vez que a própria *REST API* da *Confluent Platform* não o possui por razões de segurança. Dessa forma, numa situação que necessite de tal medida, esta operação deverá ser realizada via linha de comandos nos servidores no qual o *cluster* do *Kafka* se encontra em execução.

Assim, o funcionamento básico da *REST API* pode ser representado pelos *endpoints* definidos no âmbito desta dissertação e apresentados de seguida. A *API* possuirá ainda um mecanismo

de autenticação de utilizadores desenvolvido através de *JSON Web Tokens* (*JWT*), permitindo assim manter o nível de segurança no acesso à *API*.

De referir que nas tabelas seguintes, o texto apresentado entre os seguintes símbolos "{}" representa os parâmetros configuráveis para cada *endpoint*.

Na Tabela 13 são representados os *endpoints* das operações relativas à gestão de utilizadores.

Tabela 13: *Access System REST API - Users endpoints*

Métodos	**URL**	**Descrição**
GET	/users	Listar todos os *users*
POST	/users	Registar um *user*
POST	/auth	Autenticar um *user*
GET	/users/{userId}	Informação de um *user*
PUT	/users/{userId}	Editar um *user*
DELETE	/users/{userId}	Eliminar um *user*
GET	/users/roles	Listar todos os *roles*

Na Tabela 14 é apresentado o único *endpoint* com informação do *cluster* do *Kafka*, permitindo assim monitorizar o *cluster* e a disponibilidade do mesmo.

Tabela 14: *Access System REST API - Brokers endpoints*

Métodos	**URL**	**Descrição**
GET	/brokers	Listar todos os *brokers* do *cluster*

Na Tabela 15 são apresentados os *endpoints* relacionados com as instalações e os tópicos, são também apresentados os *endpoints* que permitem o registo e leitura de mensagens no *Kafka*.

Tabela 15: *Access System REST API – Sites/Topics endpoints*

Métodos	**URL**	**Descrição**
GET	/sites	Listar todas a instalações
POST	/sites	Criar uma instalação
GET	/sites/{siteId}	Informação de uma instalação
PUT	/sites/{siteId}	Editar uma instalação
DELETE	/sites/{siteId}	Eliminar uma instalação
POST	/sites/{siteId}/users	Associar utilizadores a uma instalação
GET	/sites/{siteId}/topics	Listar todos os tópicos
GET	/sites/{siteId}/topics/{topicId}	Informação de um tópico
POST	/sites/{siteId}/topics	Criar um tópico
GET	/sites/{siteId}/topics/{topicId}/messages	Listar todas a mensagens de um tópico
GET	/sites/{siteId}/topics/{topicId}/messages/{key}	Informação de uma mensagem
POST	/sites/{siteId}/topics/{topicId}/messages	Registar um conjunto de mensagens

Para a listagem de mensagens de determinado tópico existe ainda a possibilidade de definir parâmetros de pesquisa, que facilitam a recolha e seleção de informação, como é exemplificado na Tabela 16.

Tabela 16: *Access System REST API* – Exemplos de filtros de mensagens

Exemplos	**Descrição**
user=1234	Listar as mensagens registadas pelo utilizador com o identificador 1234
timestamp=1540254230041&interval=2592000000	Listar mensagens registadas num período de 30 dias, com início no dia 23/10/2018 às 01:23:50:0041 horas
decrypt=true	Listar todas as mensagens com o seu conteúdo desencriptado
user=1234&timestamp=1540254230041&interval=2592000000&decrypt=true	Listar as mensagens com o seu conteúdo desencriptado e registadas pelo utilizador com o identificador 1234 num período de 30 dias, com início no dia 23/10/2018 às 01:23:50:0041 horas

5.2.3 *Web App* e *Web App DB*

A aplicação *web* (*Web App*) foi desenvolvida em conjunto com o meu colega Diogo Vigo sobre um projeto modelo já utilizado pela empresa, o que agilizou o desenvolvimento e facilita a manutenção futura. Assim, a aplicação foi desenvolvida em *.NET* através do padrão *MVC*, permitindo assim separar responsabilidades e lógicas de negócio da camada de apresentação de dados, juntamente foi utilizado o *Entity Framework* para mapeamento das entidades da base de dados.

Para armazenamento dos dados utilizou-se uma base de dados *MySQL* (*Web App DB*) e a interface gráfica foi desenvolvida para se adaptar a qualquer dispositivo. Desta forma é possível fornecer ao cliente uma plataforma com funcionalidades de gestão de instalações e tópicos, e uma *dashboard* para gestão das mensagens registadas no *Kafka*, bem como todas as funcionalidades associadas à plataforma *IoT*.

5.2.3.1 Modelo de dados

O modelo de dados da *Web App*, estruturado em conjunto com o meu colega Diogo Vigo, no qual as entidades são armazenadas na *Web App DB* é representado na Figura 21 pelas seguintes entidades:

- ***ASPNetUser*** – Representa um utilizador do sistema. E. g. Administrador do sistema;
- ***ASPNetRole*** – Representa uma função de utilizador do sistema. E. g. *SuperAdmin*;
- ***Company*** – Representa uma empresa. E. g. Empresa Teste;
- ***URL*** – Representa um *URL* utilizado para comunicar com serviços externos. E. g. *URL* utilizado para comunicar com a *REST API* do *Access System*;
- ***Site*** – Representa uma instalação. E. g. Armazém 3 da empresa Teste;

- ***Topic*** – Representa um tópico do *Kafka*. E. g. Registo de dados críticos;
- ***Device*** – Representa um dispositivo físico. E. g. *Raspberry pi zero*;
- ***DeviceType*** – Identifica o tipo de dispositivo. E. g. *ESP32*;
- ***Sensor*** – Representa o tipo de sensor. E. g. Temperatura;
- ***Alert*** – Representa um alerta que é despoletado caso os valores captados por um sensor não se encontrem dentro dos intervalos definidos. E. g. Temperatura crítica;
- ***TriggeredAlert*** – Representa um alerta já despoletado. E. g. Temperatura ultrapassou os 50 °C dia 10/03/2019 às 14:10:23 horas;
- ***Action*** – Representa uma ação para um alerta. E. g. Registar ocorrência no *Kafka*.

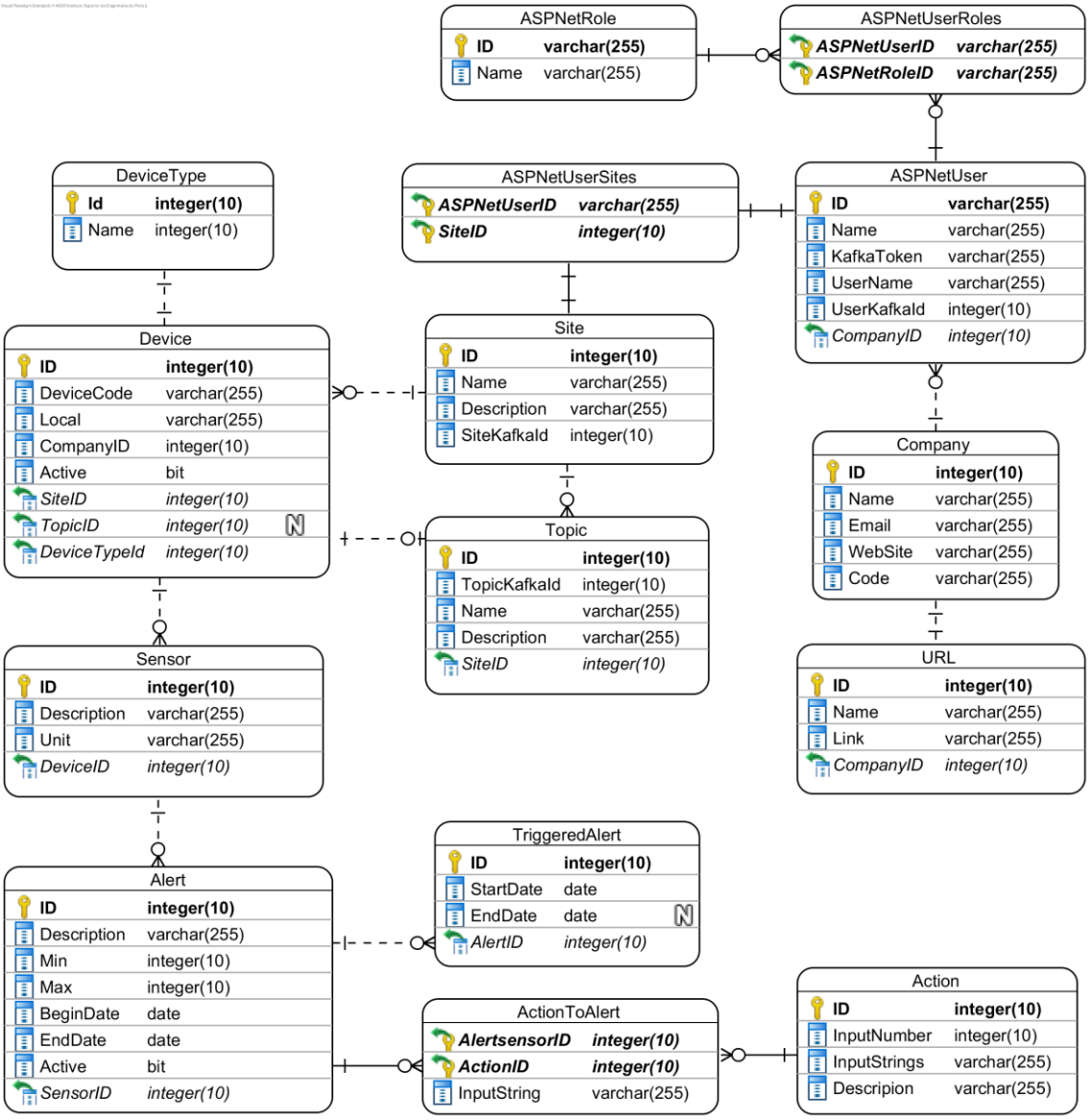


Figura 21: Modelo de dados da *Web App*

5.2.3.2 Casos de uso

A *Web App* foi desenvolvida essencialmente para consulta e gestão dos dados armazenados, no caso do produto *inCloud for Safemed* são os dados obtidos pelos sensores do Sistema *IoT* e considerados como críticos. Foram também desenvolvidas funcionalidades para gestão do Sistema *IoT*, contudo apenas são apresentadas neste documento as funcionalidades desenvolvidas para no âmbito desta dissertação. Assim, e de acordo com a Figura 22, os casos de uso da *Web App* para cada função são os seguintes:

- ***SuperAdmin*** – Função com autorização para realizar todas as operações da aplicação;
- ***CompanyAdmin*** – Função com autorização para realizar *login* e *logout*, alterar a sua *password* e, gerir todos as instalações e tópicos que seja proprietário;
- ***CompanyUser*** – Função com autorização para realizar *login* e *logout*, alterar a sua *password*, listar as instalações que seja proprietário e gerir os tópicos que também seja proprietário.

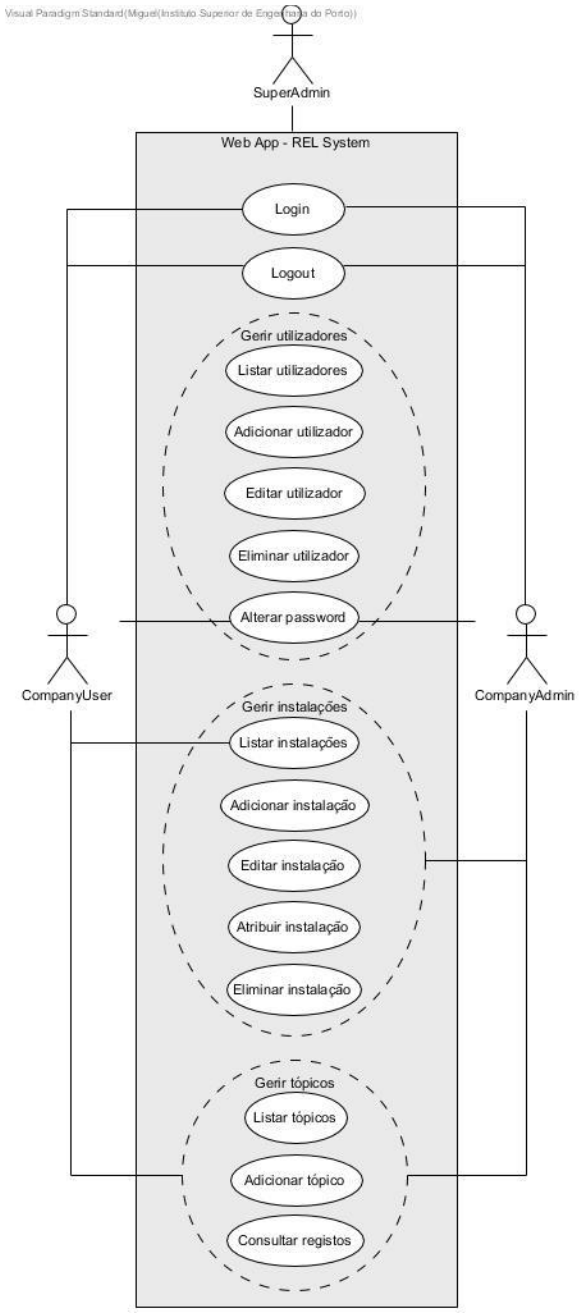


Figura 22: Diagrama de casos de uso da *Web App*

5.2.4 Fluxo de processos

Para uma melhor compreensão do fluxo dos processos que ocorrem entre os vários componentes do sistema, são apresentados os diagramas de atividades das duas operações fundamentais da solução, sendo elas, o registo e consulta de mensagens. O registo de mensagens, representado na Figura 23, ocorre de forma automática quando um alerta é despoletado e o mesmo possui a ação de armazenamento no *Kafka*.

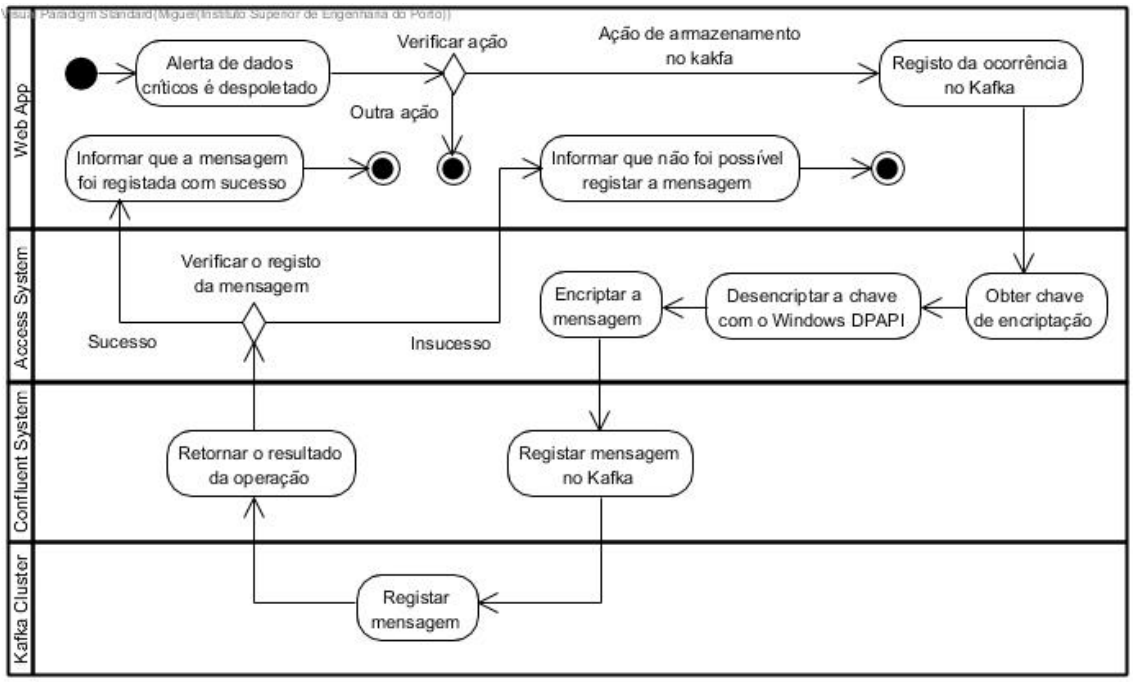


Figura 23: Diagrama de atividades – Registar Mensagem

Na Figura 24, é representada o caso de uso de consulta de registos num tópico, sendo que para um utilizador realizar essa operação, necessita de direitos de acesso à instalação ao qual o tópico está associado.

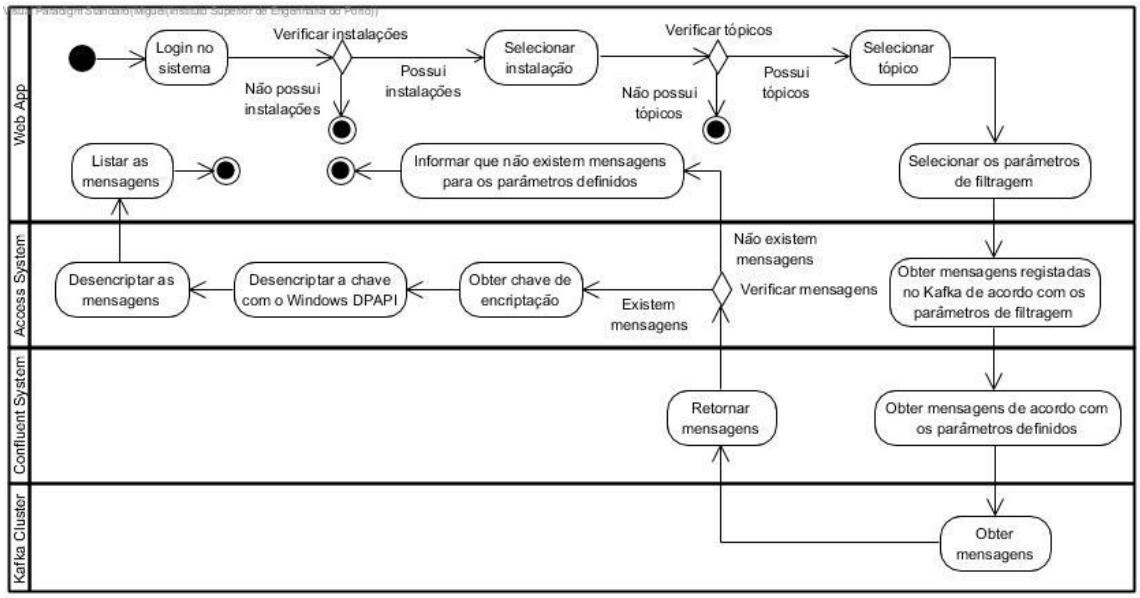


Figura 24: Diagrama de atividades – Consultar registos

5.2.5 Prova de conceito

Dadas as limitações de equipamento disponibilizado pela empresa, a arquitetura foi implementada sobre apenas dois servidores, como é apresentado na Figura 25, no entanto todos os componentes foram implementados praticamente como proposto na arquitetura definida. Sendo assim, a implementação da arquitetura do sistema como prova de conceito foi estruturada da seguinte forma:

- ***Access System & Web App Server*** – Servidor Windows para execução dos componentes relativos ao *Access System* e dos componentes relativos à *Web App*;
- ***Kafka & Confluent Server*** – Servidor *Linux* disponibilizado pela empresa e localizado nas instalações na mesma, responsável por executar os componentes relativos à *Confluent Platform* e os componentes relativos ao *cluster* de três nós do *Apache Kafka*. Difere da arquitetura definida apenas no protocolo de comunicação, uma vez que utiliza o protocolo *HTTP* e não o protocolo *HTTPS*.

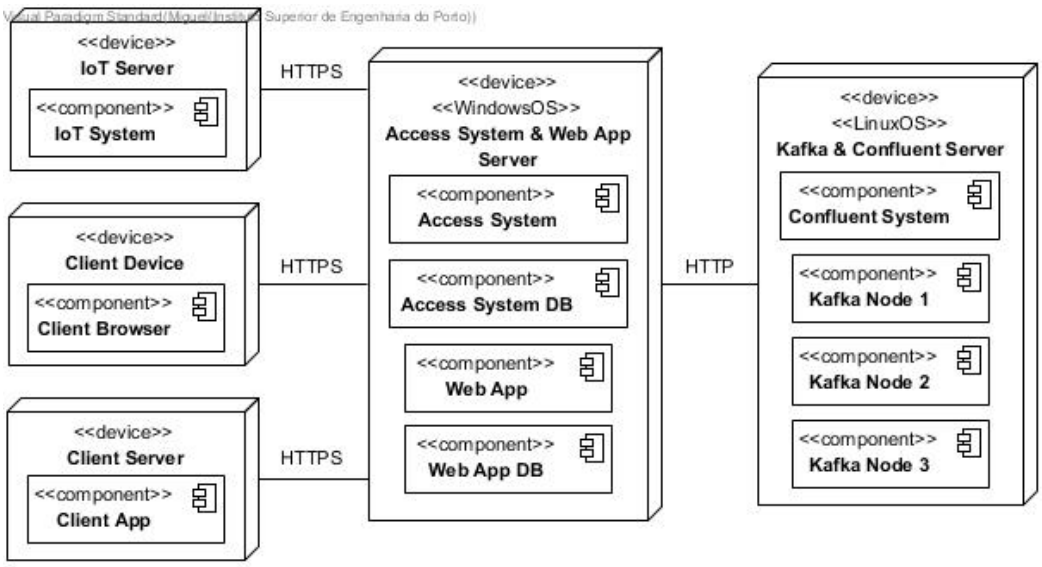


Figura 25: Diagrama de implementação da prova de conceito

5.3 Arquiteturas alternativas

Nesta secção são apresentadas as arquiteturas alternativas à arquitetura detalhada na secção 5.2, sendo que ambas as alternativas foram brevemente descritas e comparadas com a arquitetura selecionada na secção 5.1.

5.3.1 *Kafka core*

Uma das alternativas à arquitetura definida é um sistema construído sobre *Kafka core*, isto é, a tecnologia *Apache Kafka* na sua versão original e simplificada, como é apresentado numa visão de alto nível na Figura 26.

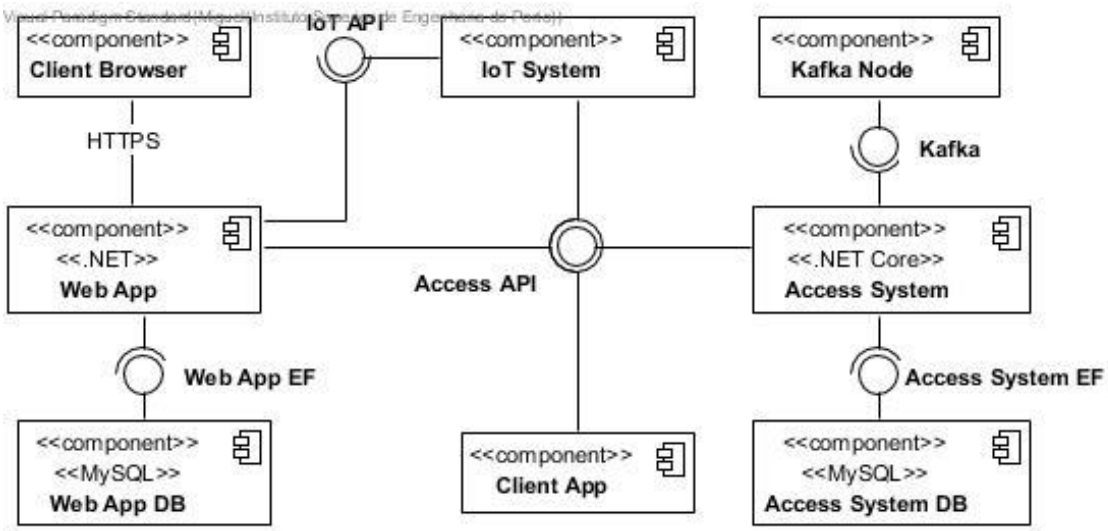


Figura 26: Diagrama de componentes de alto nível da solução alternativa com *Kafka core*

Tal como é apresentada na Figura 27, comparativamente à arquitetura definida, o componente *Access System* encontra-se inalterado, sendo que a grande diferença é não existir o componente *Confluent System*.

Assim, o *Access System* apenas consegue comunicar com o *Kafka* através do *.Net Client*, seja para a escrita ou leitura de dados, ao contrário da arquitetura definida. Desta forma, a arquitetura simplifica-se, não existindo qualquer componente extra relativo ao *Kafka* além do *Kafka Broker* e o *Zookeeper Node*.

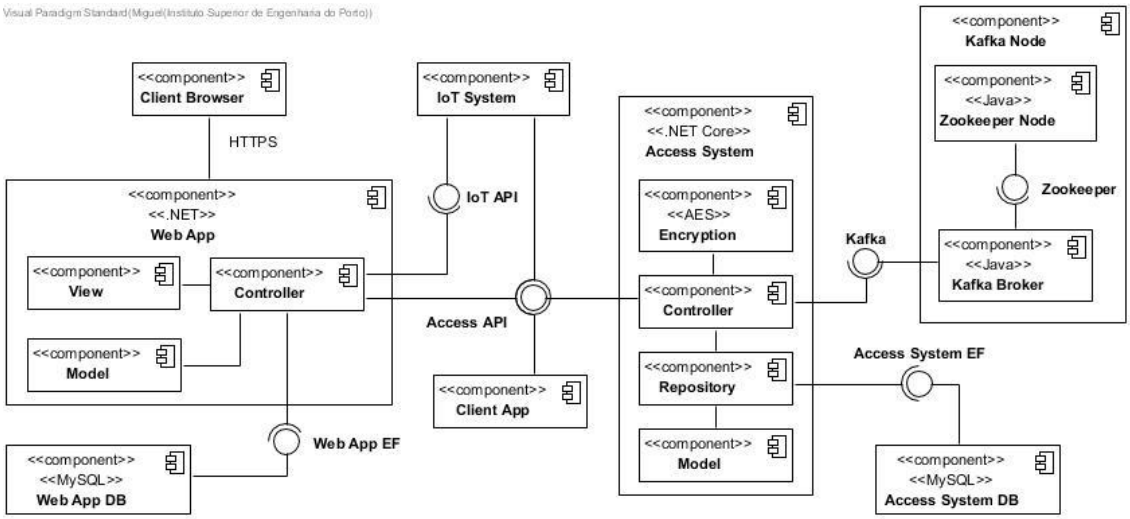


Figura 27: Diagrama de componentes de baixo nível da solução alternativa com *Kafka core*

A implementação da solução, apresentada na Figura 28, também é simplificada, uma vez que não é necessário tantos servidores como na arquitetura definida, no entanto todos os outros aspetos da solução mantêm-se.

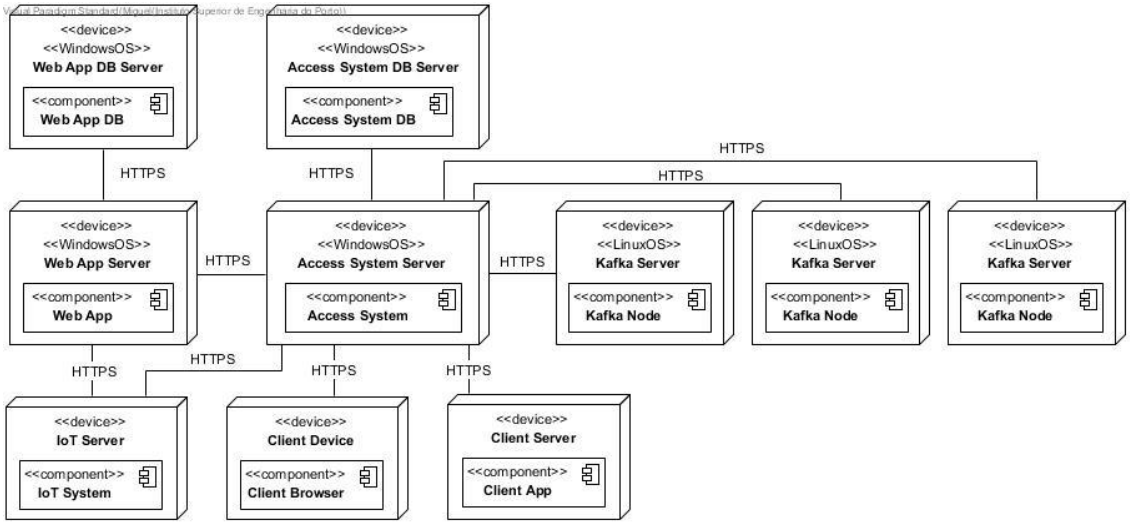


Figura 28: Diagrama de implementação da solução alternativa com *Kafka core*

5.3.2 *Kafka core* integrado com *Couchbase*

Outra alternativa à arquitetura definida é um sistema construído sobre *Kafka core*, tal como a alternativa exposta na secção 5.3.1, contudo é integrada a base de dados *Couchbase*, como é apresentado na Figura 29.

A integração de uma base de dados distribuída e imutável, permite que os dados sejam armazenados de forma estruturada e que seja possível executar *queries* complexas sobre os dados. Assim, ao contrário das outras arquiteturas apresentadas os dados são armazenados na base de dados e não no próprio *Kafka*.

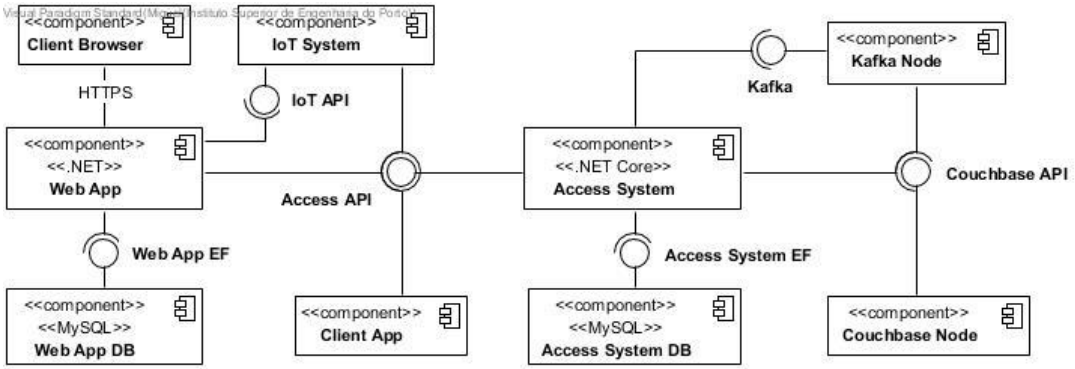


Figura 29: Diagrama de componentes de alto nível da solução alternativa com *Kafka core* integrado com *Couchbase*

Assim, comparativamente à arquitetura apresentada na secção anterior e como é representado na Figura 30, o sistema necessita de um componente para a integração do *Couchbase* (*Couchbase Node*). O *Couchbase Node* representa um nó do *Cluster* do *Couchbase* e é definido por um componente responsável por expor uma *API* para registo e leitura de dados (*Data Manager*), e um componente de gestão do *Cluster* do *Couchbase* (*Cluster Manager*).

Além disso, o sistema necessita ainda de um componente (*Kafka Connect*) para ligar o *Kafka* ao *Couchbase* e permitir desta forma que os dados introduzidos no *Kafka* sejam redirecionados automaticamente para o *Couchbase*. Por outro lado, a leitura de dados é efetuada diretamente sobre o *Couchbase*, de modo a ter acesso aos mecanismos de *query* da tecnologia.

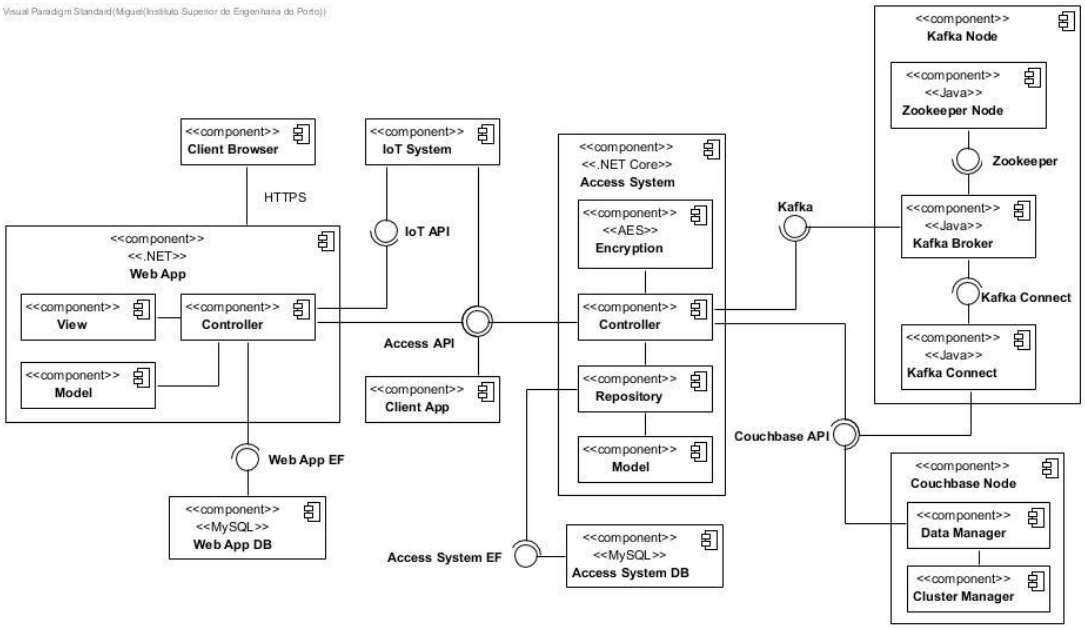


Figura 30: Diagrama de componentes de baixo nível da solução alternativa com *Kafka core* integrado com *Couchbase*

Na Figura 31 é apresentada a arquitetura de implementação do sistema, requerendo mais servidores e a instalação de mais componentes relativamente à arquitetura definida.

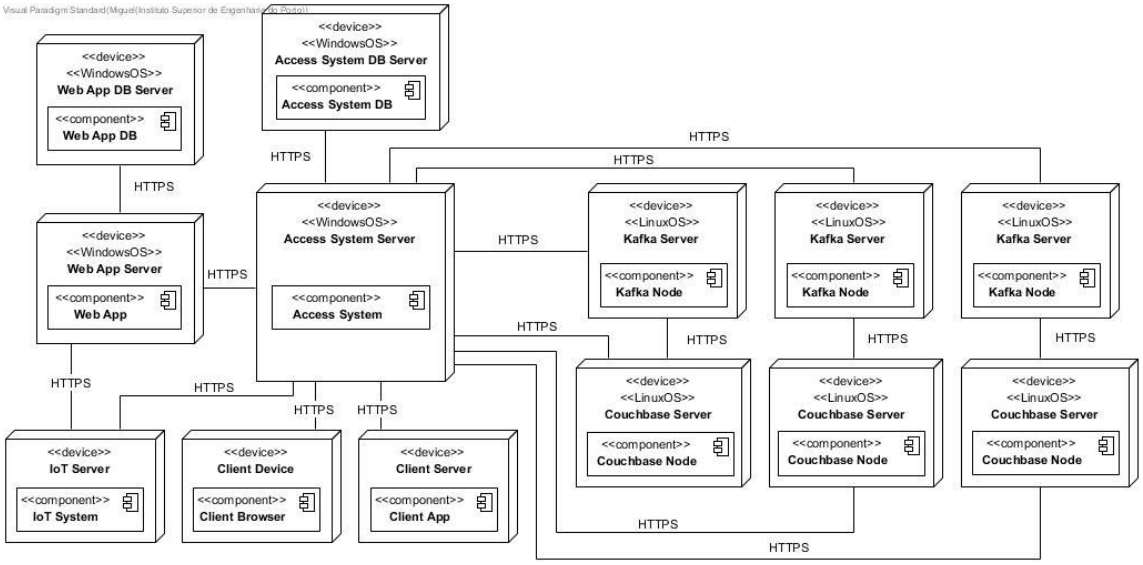


Figura 31: Diagrama de implementação da solução alternativa com *Kafka core* integrado com *Couchbase*

6 Desenvolvimento

O presente capítulo apresenta todo o processo de desenvolvimento do sistema representado no capítulo anterior e baseado na arquitetura definida como prova de conceito (secção 5.2.5). É assim descrito o desenvolvimento dos componentes referentes ao *Kafka*, dos componentes relativos ao *Access System* e por fim todo o desenvolvimento da *Web App*. São também utilizados alguns excertos de código para expor o processo de desenvolvimento, nos quais algumas partes se encontram comentadas dada a sua extensão.

6.1 *Kafka Node* e *Confluent System*

Tal como foi apresentado na arquitetura da prova de conceito, os componentes *Kafka Node* e *Confluent System* foram implementados num só servidor, sendo desta forma possível configurar ambos em simultâneo.

Para a configuração dos respetivos componentes utilizou-se ficheiros *YML* (baseado na linguagem *YAML*) para configurar imagens *docker* disponibilizadas pela *Confluent Platform*. Os ficheiros de configuração são executadas pelo *Docker Compose*, uma ferramenta que permite executar aplicações *docker multi-container*, possibilitando assim a execução dos vários componentes em simultâneo.

Desta forma, o ficheiro de configuração dos componentes relativos ao *Kafka* (*Kafka Node* e *Zookeeper Node*) e dos componentes referentes ao *Confluent System* (*Kafka REST* e *KSQL Server*) podem ser configurados num ficheiro com a estrutura apresentada no Código 1:

- ***Zookeeper*** – Configuração dos três componentes *Zookeeper Node*;
- ***Kafka*** - Configuração dos três componentes *Kafka Node*;
- ***REST Proxy*** - Configuração do módulo de acesso via *REST API* da *Confluent Platform* (componente *Kafka REST*);
- ***KSQL Server*** - Configuração do módulo para execução de *queries SQL* da *Confluent Platform* (componente *KSQL Server*).

```
---
version: '2'
services:
  zookeeper-1: #Zookeeper Node 1 configuration
  zookeeper-2: #Zookeeper Node 2 configuration
  zookeeper-3: #Zookeeper Node 3 configuration
  kafka-1: #Kafka Node 1 configuration
  kafka-2: #Kafka Node 2 configuration
  kafka-3: #Kafka Node 3 configuration
  rest-proxy: #Kafka REST configuration
  ksql-server: #KSQL Server configuration
```

Código 1: Estrutura do ficheiro de configuração *YML*

Ao contrário do que foi apresentada na arquitetura definida e como já foi referido na arquitetura da prova de conceito, não se utilizou o protocolo de comunicação *HTTPS* uma vez que não foi possível configurar nenhum certificado *SSL*/*TLS*. Desta forma, todas as comunicações realizadas com os componentes do *Kafka Node* e do *Confluent System* utilizam o protocolo *HTTP*.

6.1.1 *Kafka* e *Zookeeper*

A configuração do *cluster* do *Zookeeper* baseia-se essencialmente em cinco aspetos de configuração e é representada pelo Código 2:

- ***Image*** - Imagem que utiliza para criar o contentor no qual será executado;
- ***Ports*** – Mapeamento das portas disponíveis para acesso externo, das quais outros nós do *Zookeeper*, nós do *Kafka* ou outros componentes podem utilizar para comunicar;
- ***ZOOKEEPER_SERVER_ID*** – Identificador do nó do *Zookeeper*;
- ***ZOOKEEPER_CLIENT_PORT*** – Identificador da porta à qual o *cluster* do *Kafka* se vai ligar;
- ***ZOOKEEPER_SERVERS*** – Identificação do *cluster* do *Zookeeper*.

```
zookeeper-1:
  image: confluentinc/cp-zookeeper
  hostname: zookeeper-1
  container_name: zookeeper-1
  ports:
    - "12181:12181"
    - "12888:12888"
    - "13888:13888"
  environment:
    ZOOKEEPER_SERVER_ID: 1
    ZOOKEEPER_CLIENT_PORT: 12181
    ZOOKEEPER_TICK_TIME: 2000
    ZOOKEEPER_INIT_LIMIT: 5
    ZOOKEEPER_SYNC_LIMIT: 2
    ZOOKEEPER_SERVERS: zookeeper-1:12888:13888;zookeeper-
2:22888:23888;zookeeper-3:32888:33888

zookeeper-2:
  image: confluentinc/cp-zookeeper
  hostname: zookeeper-2
  container_name: zookeeper-2
  ports:
    - "22181:22181"
    - "22888:22888"
    - "23888:23888"
  environment:
    ZOOKEEPER_SERVER_ID: 2
    ZOOKEEPER_CLIENT_PORT: 22181
    ZOOKEEPER_TICK_TIME: 2000
    ZOOKEEPER_INIT_LIMIT: 5
    ZOOKEEPER_SYNC_LIMIT: 2
    ZOOKEEPER_SERVERS: zookeeper-1:12888:13888;zookeeper-
2:22888:23888;zookeeper-3:32888:33888

zookeeper-3:
  image: confluentinc/cp-zookeeper
  hostname: zookeeper-3
  container_name: zookeeper-3
  ports:
    - "32181:32181"
    - "32888:32888"
    - "33888:33888"
  environment:
    ZOOKEEPER_SERVER_ID: 3
    ZOOKEEPER_CLIENT_PORT: 32181
    ZOOKEEPER_TICK_TIME: 2000
    ZOOKEEPER_INIT_LIMIT: 5
    ZOOKEEPER_SYNC_LIMIT: 2
    ZOOKEEPER_SERVERS: zookeeper-1:12888:13888;zookeeper-
2:22888:23888;zookeeper-3:32888:33888
```

Código 2: Configuração do *cluster* do *Zookeeper*

A configuração do *cluster* do *Kafka* necessitou de algumas considerações adicionais, sendo representada essencialmente pelos seguintes atributos e pelo Código 3:

- ***Image*** - Imagem que utiliza para criar o contentor no qual será executado;
- ***Depends_on*** – Identificação dos componentes dos quais depende para ser executado;
- ***Ports*** – Mapeamento das portas disponíveis para acesso externo;

- ***KAFKA_BROKER_ID*** – Identificador do nó do *Kafka*;
- ***KAFKA_ZOOKEEPER_CONNECT*** – Identificação do *cluster* do *Zookeeper*;
- ***KAFKA_NUM_PARTITIONS*** – Número de partições por tópico;
- ***KAFKA_DEFAULT_REPLICATION_FACTOR*** – Fator de replicação para os tópicos criados.

```
1.  kafka-1:
2.    image: confluentinc/cp-kafka
3.    hostname: kafka-1
4.    container_name: kafka-1
5.    ports:
6.      - "19092:19092"
7.    depends_on:
8.      - zookeeper-1
9.      - zookeeper-2
10.     - zookeeper-3
11.   environment:
12.     KAFKA_BROKER_ID: 1
13.     KAFKA_ZOOKEEPER_CONNECT: zookeeper-1:12181,zookeeper-2:22181,zookeeper-
    3:32181
14.     KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka-1:19092
15.     KAFKA_NUM_PARTITIONS: 3
16.     KAFKA_DEFAULT_REPLICATION_FACTOR: 3
17.
18. kafka-2:
19.   image: confluentinc/cp-kafka
20.   hostname: kafka-2
21.   container_name: kafka-2
22.   ports:
23.     - "29092:29092"
24.   depends_on:
25.     - zookeeper-1
26.     - zookeeper-2
27.     - zookeeper-3
28.   environment:
29.     KAFKA_BROKER_ID: 2
30.     KAFKA_ZOOKEEPER_CONNECT: zookeeper-1:12181,zookeeper-2:22181,zookeeper-
    3:32181
31.     KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka-2:29092
32.     KAFKA_NUM_PARTITIONS: 3
33.     KAFKA_DEFAULT_REPLICATION_FACTOR: 3
34.
35. kafka-3:
36.   image: confluentinc/cp-kafka
37.   hostname: kafka-3
38.   container_name: kafka-3
39.   ports:
40.     - "39092:39092"
41.   depends_on:
42.     - zookeeper-1
43.     - zookeeper-2
44.     - zookeeper-3
45.   environment:
46.     KAFKA_BROKER_ID: 3
47.     KAFKA_ZOOKEEPER_CONNECT: zookeeper-1:12181,zookeeper-2:22181,zookeeper-
    3:32181
48.     KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka-3:39092
49.     KAFKA_NUM_PARTITIONS: 3
50.     KAFKA_DEFAULT_REPLICATION_FACTOR: 3
```

Código 3: Configuração do *cluster* do *Kafka*

Além das configurações definidas e descritas anteriormente, foram tidas em consideração algumas configurações importantes que já se encontravam definidas de origem:

- ***KAFKA_AUTO_CREATE_TOPICS_ENABLE*** – Criação automática de tópicos quando alguma mensagem é registada com a identificação de um tópico que ainda não existe, por predefinição esta operação está ativa;
- ***KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR*** – Fator de replicação dos *offsets* do tópico, por predefinição o fator de replicação é de três;
- ***KAFKA_LOG_CLEANUP_POLICY*** – Política de retenção definida quando as condições de retenção são atingidas, por predefinição elimina os dados;
- ***KAFKA_LOG_RETENTION_MS*** – Tempo de retenção dos dados em milissegundos, por predefinição os dados são retidos por sete dias (604800000 ms);
- ***KAFKA_LOG_RETENTION_CHECK_INTERVALS_MS*** – Frequência em milissegundos no qual o sistema verifica a existência de dados elegíveis para eliminação com base na política de retenção, por predefinição o valor é cinco minutos (300000 ms);
- ***KAFKA_LOG_RETENTION_BYTES*** – Condição da política de retenção através do tamanho máximo permitido, por predefinição esta condição está inativa com um valor de -1, não afetando de qualquer forma a política de retenção;
- ***KAFKA_AUTO_LEADER_REBALANCE_ENABLE*** – Eleição automática dos nós líderes no *cluster*, por predefinição esta operação está ativa e é executada com uma frequência de 5 minutos.

6.1.2 Módulos da *Confluent Platform*

A configuração do módulo do *REST Proxy* foi bastante simplificada, podendo ser representada pelos seguintes atributos de configuração e pelo Código 4:

- ***Image*** - Imagem que utiliza para criar o contentor no qual será executado;
- ***Depends_on*** – Identificação dos componentes dos quais depende para ser executado;
- ***Ports*** – Mapeamento das portas disponíveis para acesso externo;
- ***KAFKA_REST_ZOOKEEPER_CONNECT*** – Identificação do *cluster* do *Zookeeper*;
- ***KAFKA_REST_BOOTSTRAP_SERVERS*** - Identificação do *cluster* do *Kafka*;
- ***KAFKA_REST_LISTENERS*** – Identificação do *URL* de comunicação disponibilizado.

```
rest-proxy:
  image: confluentinc/cp-kafka-rest
  hostname: rest-proxy
  container_name: rest-proxy
  depends_on:
    - zookeeper-1
    - zookeeper-2
    - zookeeper-3
    - kafka-1
    - kafka-2
    - kafka-3
  ports:
    - 8082:8082
  environment:
    KAFKA_REST_HOST_NAME: rest-proxy
    KAFKA_REST_ZOOKEEPER_CONNECT: zookeeper-1:12181,zookeeper-2:22181,zookeeper-3:32181
    KAFKA_REST_BOOTSTRAP_SERVERS: kafka-1:19092,kafka-2:29092,kafka-3:39092
    KAFKA_REST_LISTENERS: "http://0.0.0.0:8082"
```

Código 4: Configuração do módulo *REST Proxy*

O módulo da *REST Proxy*, tal como já foi apresentado, permite o acesso ao *Kafka* através de uma *REST API* disponibilizada, dessa forma foram utilizados *endpoints* dessa mesma *API* para algumas operações, como é apresentado na Tabela 17.

Tabela 17: *REST Proxy API – Endpoints* utilizados [82]

Métodos	**URL**	**Descrição**
GET	/brokers	Listar *brokers* do *cluster* do *Kafka*
POST	/topics/{topicName}	Registar mensagens num tópico

Para o *endpoint* de registo de mensagens num tópico, além do parâmetro referente ao nome do tópico, é ainda necessário preencher em *JSON* o conteúdo (*body*) do pedido *HTTP* com as mensagens a registar, tal como é apresentado no Código 5.

```
{
  "records": [
    {
      "key": 1,
      "value": {"value": "2mRvZo4pUcb8BP/123LJEQ==", "user": 32452}
    },
    {
      "key": 2,
      "value": {"value": "AbjOwklv742e5N5Yqmp0ww==", "user": 54647}
    },
    {
      "key": 3,
      "value": {"value": "E34p41ifEKl+Z02/fMLnaQ==", "user": 93244}
    }
  ]
}
```

Código 5: *Body* do pedido *HTTP* para registo de mensagens

O módulo do *KSQL Server* foi configurado de forma idêntica à do *REST Proxy*, podendo ser apresentada pelas propriedades seguintes e pelo Código 6:

- ***Image*** - Imagem que utiliza para criar o contentor no qual será executado;
- ***Depends_on*** – Identificação dos componentes dos quais depende para ser executado;
- ***Ports*** – Mapeamento das portas disponíveis para acesso externo;
- ***KSQL_BOOTSTRAP_SERVERS*** - Identificação do *cluster* do *Kafka*;
- ***KSQL_LISTENERS*** – Identificação do *URL* de comunicação disponibilizado.

```
1.  ksql-server:
2.    image: confluentinc/cp-ksql-server
3.    hostname: ksql-server
4.    container_name: ksql-server
5.    depends_on:
6.      - kafka-1
7.      - kafka-2
8.      - kafka-3
9.    ports:
10.     - "8088:8088"
11.   environment:
12.     KSQL_HOST_NAME: ksql-server
13.     KSQL_CONFIG_DIR: "/etc/ksql"
14.     KSQL_LOG4J_OPTS: "-Dlog4j.configuration=file:/etc/ksql/log4j-
    rolling.properties"
15.     KSQL_BOOTSTRAP_SERVERS: kafka-1:19092,kafka-2:29092,kafka-3:39092
16.     KSQL_LISTENERS: "http://0.0.0.0:8088"
```

Código 6: Configuração do módulo *KSQL Server*

O módulo *KSQL Server* é utilizado essencialmente como mecanismos de acesso aos dados armazenados, porém, e como já foi referido anteriormente, é necessário a criação de *streams* para permitir o mapeamento desses mesmos dados através de *JSON*. Uma vez que o *KSQL Server* disponibiliza uma *REST API*, todas as operações são realizadas por esse meio. Assim, tal como é apresentado na Tabela 18, são utilizados apenas dois *endpoints* da *API*, um para a criação das *streams* e listagem de todas as mensagens de um tópico sem qualquer filtro, e outro para a execução de *queries* através das *streams* criadas, o que permite a filtragem de mensagens. Para ambos os *endpoints* é utilizada a linguagem *SQL* para execução das operações.

Tabela 18: *KSQL REST API* – *Endpoints* utilizados [83]

Métodos	URL	Descrição
POST	/ksql	Executar operações de gestão de *streams* e listagem de mensagens
POST	/query	Executar *queries SQL* via *streams* sobre os dados armazenados

Tal como foi apresentado para o *endpoint* de registo de mensagens da *REST Proxy API*, também os *endpoints* da tabela anterior necessitam de conteúdo *JSON* no seu pedido *HTTP*, de forma a identificar a operação a executar. Assim, para a criação de *streams* é realizado o pedido conforme os dados apresentados no Código 7.

```
{
  "ksql": "CREATE STREAM streamName (value VARCHAR, user BIGINT) WITH (KAFKA_TOPIC='topicName', VALUE_FORMAT='JSON');",
  "streamsProperties": {
    "ksql.streams.auto.offset.reset": "earliest"
  }
}
```

Código 7: *Body* do pedido *HTTP* para criação de *streams*

O Código 8 representa a operação de listagem de todas as mensagens de determinado tópico desde a sua mensagem mais antiga, e uma vez que não é aplicado qualquer mapeamento através de *streams* ou qualquer filtragem, a operação é bastante eficiente.

```
{
  "ksql": "PRINT 'topicName' FROM BEGINNING;",
  "streamsProperties": {
    "ksql.streams.auto.offset.reset": "earliest"
  }
}
```

Código 8: *Body* do pedido *HTTP* para listagem de todas as mensagens de um tópico

A listagem e filtragem de mensagens é possível através da combinação da linguagem *SQL* com o mapeamento realizado pelas *streams*, tal como é apresentado no Código 9.

```
{
  "ksql": "SELECT * FROM streamName WHERE ROWTIME>=1540254230040 AND ROWTIME<1540254230090 AND USER=54647;",
  "streamsProperties": {
    "ksql.streams.auto.offset.reset": "earliest"
  }
}
```

Código 9: *Body* do pedido *HTTP* para listagem e filtragem de mensagens

A propriedade *ksql.streams.auto.offset.reset* apresentada em todo os pedidos descritos anteriormente identifica a forma como as mensagens são tratadas, neste caso, como o valor da propriedade está definido como *earliest*, as mensagens são mapeadas e retornadas a partir da mensagem mais antiga, caso contrário são obtidas à medida que as mesmas são registadas. Esta propriedade é necessária uma vez que os pedidos relativos à listagem de mensagens retornam a informação de uma forma particular, através do *chunked transfer encoding*, um mecanismo de transferência de dados em *stream* presente no protocolo *HTTP*.

6.2 *Access System*

O desenvolvimento do *Access System* ocorreu tal como proposto na arquitetura, sendo desenvolvida uma *REST API* em *.NET Core* (C#) na qual aplicações externas, tal como a *Web App*,

podem realizar operações sobre os componentes do *Kafka*. A *API* foi configurada com o protocolo de comunicação *HTTPS*, adicionando assim uma camada de segurança ao sistema.

Durante o desenvolvimento surgiu a necessidade de utilizar as *frameworks* apresentados de seguida de forma facilitar o desenvolvimento de determinadas operações, sendo que todos os *frameworks* são disponibilizados pelo *NuGet*, o *package manager* para *.NET*:

- ***.NET Core*** – *Framework* para desenvolvimento de aplicações;
- ***Entity Framework Core*** - Mapeamento das entidades armazenadas na base de dados *MySQL* através da abordagem *code-first*;
- ***Newtonsoft JSON*** – *Framework* utilizado para mapeamento de informação em *JSON*;
- ***JWT*** - Mecanismo de autenticação através de *JSON Web Tokens*;
- ***AutoMapper*** – Mapeamento entre objetos;
- ***RestSharp*** – Cliente utilizado para a maioria dos pedidos *HTTP REST* realizados;
- ***Swashbuckle*** – Documentação detalhada da *REST API* através do *Swagger*;
- ***Bouncy Castle*** – Biblioteca de encriptação, utilizada para execução do algoritmo *AES* no modo *CTR*;
- ***xUnit*** - Biblioteca para desenvolvimento de testes unitários por classe;
- ***Moq*** - Biblioteca de suporte aos testes unitários para simulação de classes;
- ***HTTP Mock*** - Biblioteca de suporte aos testes unitários para simulação de serviços *HTTP*, permitindo simular os serviços da *Confluent Platform*;
- ***Serilog*** – Permite o registo de *logs* por níveis.

Toda a *API* foi documentada através de uma estrutura em *XML* (*Extensible Markup Language*), o que permitiu, em conjunto com a *framework Swashbuckle*, gerar uma interface *web* com toda a documentação dos *endpoints* da *REST API* e a possibilidade de utilizar esses mesmos *endpoints* de forma simplificada através da interface, tal como é apresentado na Figura 39 em anexo.

Todos os processos envolvendo estruturas de dados em *JSON* utilizam o *framework Newtonsoft JSON* para o mapeamento desses mesmos dados.

Implementou-se ainda um mecanismo de registo de *logs* por níveis, através da *framework Serilog*, no qual foi definido como mínimo o nível *Information*, que de acordo com a documentação do *framework* representa eventos que ocorrem no sistema e que correspondem às suas responsabilidades e funções, normalmente são ações observáveis que o sistema pode executar. [84]

Além do nível mínimo definido, todos os níveis superiores também são registados, além disso implementou-se ainda uma classe intermédia para registo dos *logs* relativos aos pedidos *HTTP* realizados à API, permitindo assim ter um maior controlo das operações executadas. Todos os *logs* são registados num ficheiro de texto local gerado para cada dia.

6.2.1 Modelo de dados

Para o desenvolvimento do modelo de dados apresentado na secção 5.2.2.1 utilizou-se a abordagem *code-first* com o auxílio do *Entity Framework Core*, sendo assim possível gerar todo o modelo de dados na base de dados *MySQL* com base nas classes modelo de cada entidade.

Uma vez que a *API* desenvolvida requeria o envio e receção de informação, e muitas das vezes não era adequada a transferência da totalidade de informação de uma entidade, implementou-se várias classes *DTO* (*Data Transfer Object*) estruturadas à medida do que era necessário. Assim, para facilitar todo o processo de mapeamento entre classes modelo e classes *DTO* utilizou-se o *framework AutoMapper*.

Utilizou-se também o padrão *Repository*, no qual se criou uma classe *interface* para cada classe *repository*, permitindo criar uma camada de abstração entre a camada do modelo de negócio e a camada de acesso aos dados. Além disso, permite ainda a criação e utilização de classes *repository* simuladas nos testes unitários.

6.2.2 Autenticação e autorização

O mecanismo de autenticação do sistema foi desenvolvido com base em *JSON Web Tokens* e no sistema de identidades presente no *.NET*. Assim, a produção de *tokens* para cada utilizador, que incluem o seu identificador (*ID*) e a sua função no sistema (*Role*), foi implementada tal como é apresentada no Código 10. O *JWT* é gerado através do segredo armazenado no ficheiro de configuração *appsettings.json* e tem uma validade de uma semana.

```
1.  public static String GenerateJwt(User user, AppSettings appSettings)
2.  {
3.      var tokenHandler = new JwtSecurityTokenHandler();
4.      var key = Encoding.ASCII.GetBytes(appSettings.Secret);
5.      var tokenDescriptor = new SecurityTokenDescriptor
6.      {
7.          Subject = new ClaimsIdentity(new Claim[]
8.          {
9.              new Claim(ClaimTypes.Name, user.Id.ToString()),
10.             new Claim(ClaimTypes.Role, user.Role)
11.         }),
12.         Expires = DateTime.UtcNow.AddDays(7), // 1 Week
13.         SigningCredentials = new SigningCredentials(new SymmetricSecurityKe
    y(key), SecurityAlgorithms.HmacSha256Signature)
14.     };
15.     var token = tokenHandler.CreateToken(tokenDescriptor);
16.     return tokenHandler.WriteToken(token);
17. }
```

Código 10: Função de produção de *JSON Web Tokens*

Tal como apresentado na secção 5.2.2.1, criaram-se três funções (*Roles*) no sistema, o *SuperAdmin*, o *CompanyAdmin* e o *CompanyUser*, permitindo assim mecanismos de autorização nas operações da *API*, sendo que, para limitar o acesso a determinado *endpoint* utilizou-se o atributo *Authorize* presente no *.NET*.

6.2.3 Operações sobre o *Kafka* e *Confluent Platform*

Tal como foi descrito nas secções 5.2.1 e 6.1.2, são apenas utilizados dois *endpoints* da *REST Proxy API* e dois da *KSQL REST API* para a execução das cinco operações sobre o *Kafka* e o *Confluent Platform*. O referido é possível, uma vez que o *endpoint* de registo de mensagens, além da funcionalidade básica de registar mensagens num tópico, possibilita também a criação de um novo tópico caso o mesmo ainda não exista. Assim, dada a configuração predefinida no *cluster* do *Kafka*, é possível a criação automática de um novo tópico apenas com o registo de uma mensagem, como é representado no Código 11.

```
1.  private List<NewMessageDto> AddMessage(Topic topic, int keyCount, List<stri
    ng> messagesList)
2.  {
3.      List<NewMessageDto> newMessagesList = new List<NewMessageDto>();
4.
5.      // Setup Request
6.      var client = new RestClient(API_URL);
7.      var request = new RestRequest("topics/" + topic.Name, Method.POST);
8.      // Headers
9.      request.AddHeader("Accept", "application/vnd.kafka.v2+json, application
    /vnd.kafka+json, application/json");
10.     request.AddHeader("Content-
    Type", "application/vnd.kafka.json.v2+json");
11.     // Get AES Key
12.     AesKey aesKey = topic.Site.AesKey;
13.     // Request Body
14.     var messagesListAux = new List<object>();
15.
16.     // Format messages to JSON
17.     // ... code ...
18.     // Format messages to JSON
19.
20.     var postData = JsonConvert.SerializeObject(new { records = messagesList
    Aux });
21.     request.AddParameter("undefined", postData, ParameterType.RequestBody);
22.
23.     IRestResponse response = client.Execute(request);
24.
25.     if (response.StatusCode != HttpStatusCode.OK)
26.     {
27.         return null;
28.     }
29.
30.     return newMessagesList;
31. }
```

Código 11: Função de registo de mensagens num tópico do *Kafka*

O excerto de código apresentado anteriormente representa um pedido *HTTP* realizado através do *framework RestSharp*, no qual o código comentado com "*Format messages to JSON*" representa a formatação das mensagens com uma estrutura *JSON* idêntica à apresentada no Código 5.

No Código 12 está representado a função responsável pelo pedido *HTTP* à *REST API* do *KSQL Server* para criação de uma *stream* estruturada em *JSON* e associada a determinado tópico. A função referida é executada imediatamente após a criação do tópico.

```
1.  private bool CreateStream(Topic topic)
2.  {
3.      string topicName = topic.Name;
4.      string streamName = topic.StreamName;
5.
6.      // Setup Request
7.      var client = new RestClient(KSQL_API_URL);
8.      var request = new RestRequest("ksql", Method.POST);
9.      // Headers
10.     request.AddHeader("Accept", "application/vnd.ksql.v1+json");
11.     request.AddHeader("Content-Type", "application/vnd.ksql.v1+json");
12.     // Request Body
13.     string postData = "{ \"ksql\": \"CREATE STREAM " + streamName + " (valu
    e VARCHAR, user BIGINT) WITH (KAFKA_TOPIC='" + topicName + "', VALUE_FORMAT
    ='JSON');\", \"streamsProperties\": { \"ksql.streams.auto.offset.reset\": \
    "earliest\" } }";
14.
15.     request.AddParameter("undefined", postData, ParameterType.RequestBody);
16.     IRestResponse response = client.Execute(request);
17.
18.     if (response.StatusCode != HttpStatusCode.OK)
19.     {
20.         return false;
21.     }
22.
23.     return true;
24. }
```

Código 12: Função de criação de *streams* no *KSQL Server*

Como já foi referido anteriormente, a listagem de mensagens retorna a informação através do mecanismo *chunked transfer encoding*, sendo assim necessário a utilização da classe *WebRequest* presente no *.NET*, como apresentado no Código 13, uma vez que o *framework RestSharp* não permite tratar dados em *stream*. O código comentado com "*Messages from JSON to MessageDto*" representa a conversão das mensagens recebidas em *JSON* para objetos da classe *DTO MessageDto*, sendo que o conteúdo da mensagem é desencriptado conforme o valor da variável *decrypt*.

```
1.  private List<MessageDto> QueryAllMessages(Topic topic, bool decrypt)
2.  {
3.      WebRequest request = WebRequest.Create(KSQL_API_URL + "/query");
4.      request.Method = "POST";
5.      string postData = "{ \"ksql\": \"PRINT '" + topic.Name + "' FROM BEGINN
    ING;\" }";
6.      byte[] byteArray = Encoding.UTF8.GetBytes(postData);
7.      request.ContentType = "application/vnd.ksql.v1+json";
8.      request.ContentLength = byteArray.Length;
9.      Stream dataStream = request.GetRequestStream();
10.     dataStream.Write(byteArray, 0, byteArray.Length);
11.     dataStream.Close();
12.     WebResponse response = request.GetResponse();
13.
14.     if (((HttpWebResponse)response).StatusCode != HttpStatusCode.OK)
15.     {
16.         return null;
17.     }
18.
19.     Stream resStream = response.GetResponseStream();
20.     List<string> strContent = new List<string>();
21.     var reader = new StreamReader(resStream);
22.     var line = 0;
23.     while (!reader.EndOfStream)
24.     {
25.         String str = reader.ReadLine();
26.         if (line > 0)
27.         {
28.             if (str.Length == 0)
29.             {
30.                 break;
31.             }
32.             strContent.Add(str);
33.         }
34.         line++;
35.     }
36.
37.     List<MessageDto> messageDtos = new List<MessageDto>();
38.     // Messages from JSON to MessageDto
39.     // ... code ...
40.     // Messages from JSON to MessageDto
41.
42.     reader.Close();
43.     dataStream.Close();
44.     response.Close();
45.
46.     // Order messages list by key
47.     List<MessageDto> messageDtosOrderer = new List<MessageDto>();
48.     messageDtosOrderer = messageDtos.OrderBy(m => m.Key).ToList();
49.
50.     return messageDtosOrderer;
51. }
```

Código 13: Função de listagem de todas as mensagens de um tópico do *Kafka*

6.2.4 Encriptação

Todas as *passwords* associadas aos utilizadores da *API* são armazenadas num formato encriptado utilizando um método de *hashing*, utilizou-se o algoritmo *SHA-256* através de uma classe nativa do *.NET*, *SHA256Managed*, como é apresentado no Código 14.

```
1.  public static string GenerateHash(string plainText)
2.  {
3.      using (SHA256 hash = SHA256Managed.Create())
4.      {
5.          return String.Concat(hash
6.            .ComputeHash(Encoding.UTF8.GetBytes(plainText))
7.            .Select(item => item.ToString("x2")));
8.      }
9.  }
```

Código 14: Função de encriptação com o algoritmo *SHA-256*

O Código 15 representa o algoritmo de encriptação *AES* no modo *CTR* implementado através do *framework Bouncy Castle*, permitindo assim encriptar o conteúdo das mensagens armazenadas no *Kafka* com uma chave de 128 bits.

```
1.  public static string EncryptWithAES(string plainText, string keyWithDPAPI)

2.  {
3.      byte[] inputBytes = Encoding.UTF8.GetBytes(plainText);
4.      SecureRandom random = new SecureRandom();
5.      byte[] iv = new byte[16];
6.      random.NextBytes(iv);
7.      string keyStringBase64 = DecryptKeyWithDPAPI(keyWithDPAPI);
8.
9.      AesEngine engine = new AesEngine();
10.     // CTR Mode
11.     KCtrBlockCipher blockCipher = new KCtrBlockCipher(engine);
12.     PaddedBufferedBlockCipher cipher = new PaddedBufferedBlockCipher(new KC
    trBlockCipher(engine), new Pkcs7Padding());
13.     KeyParameter keyParam = new KeyParameter(Convert.FromBase64String(keySt
    ringBase64));
14.     // Generate and use IV from encryption Key
15.     ParametersWithIV keyParamWithIV = new ParametersWithIV(ParameterUtiliti
    es.CreateKeyParameter("AES", Convert.FromBase64String(keyStringBase64)), ne
    w byte[16]);
16.
17.     // Encrypt
18.     cipher.Init(true, keyParamWithIV);
19.     byte[] outputBytes = new byte[cipher.GetOutputSize(inputBytes.Length)];

20.     int length = cipher.ProcessBytes(inputBytes, outputBytes, 0);
21.     cipher.DoFinal(outputBytes, length);
22.
23.     return Convert.ToBase64String(outputBytes);
24. }
```

Código 15: Função de encriptação com o algoritmo *AES* no modo *CTR*

As chaves de encriptação de 128 bits utilizadas no algoritmo anterior são armazenadas num formato encriptado na base dados *MySQL*, garantindo assim uma medida de segurança adicional. O mecanismo responsável pela encriptação das chaves é o *Windows DPAPI* através da classe nativa do *.NET*, *ProtectedData*, assegurando assim a encriptação das chaves por máquina. O processo referido é representado pela função do Código 16.

```
1.  public static string EncryptKeyWithDPAPI(string key)
2.  {
3.      byte[] originalText = Convert.FromBase64String(key);
4.      byte[] encryptedText = ProtectedData.Protect(originalText, null, dataPr
    otectionScope);
5.
6.      return Convert.ToBase64String(encryptedText);
7.  }
```

Código 16: Função de encriptação com o *DPAPI*

A criação aleatória de chaves de 128 bits é da responsabilidade da classe *RNGCryptoServiceProvider* presente no *.NET*, tal como é apresentado no Código 17.

```
1.  public static string CreateRandomKey()
2.  {
3.      byte[] key = new byte[16];
4.      new RNGCryptoServiceProvider().GetBytes(key);
5.
6.      return Convert.ToBase64String(key);
7.  }
```

Código 17: Função de criação de chaves aleatórias de 128 bits

6.2.5 Testes unitários

Os testes unitários forma realizados para assegurar a funcionalidade das operações definidas para o *Access System*, para isso utilizou-se o *framework xUnit*, que permite a execução de testes unitários por classe. Uma vez que se implementou o padrão *Repository*, foi possível o desenvolvimento de repositórios simulados através do *framework Moq*, permitindo assim definir o conjunto de dados utilizados no decorrer dos testes.

Assim, tal como é apresentado no Código 18, o repositório simulado *MockSitesRepository* é utilizado na classe *Controller* como repositório válido, sendo assim possível realizar todos os testes ao *Controller* que envolvam esse mesmo repositório. No exemplo do código seguinte é realizado um teste unitário à função de leitura de uma instalação.

```
1.  public class SitesControllerTest
2.      {
3.          private SitesController controller;
4.          private MockSitesRepository mockSitesRepository;
5.          // ... code ...
6.
7.          public SitesControllerTest()
8.          {
9.              // Setup
10.             mockSitesRepository = new MockSitesRepository();
11.             // ... code ...
12.             controller = new SitesController(mockMapper, mockSitesRepositor
    y, mockTopicsRepository, mockUsersRepository, mockAesKeysRepository);
13.             // ... code ...
14.         }
15.
16.         [Fact]
17.         public void GetTest()
18.         {
19.             // Act
20.             var result = controller.Get(1).Result as ObjectResult;
21.
22.             // Assert
23.             var item = Assert.IsType<SiteExtendedDto>(result.Value);
24.             Assert.Equal(mockSitesRepository.GetById(1).Id, item.Id);
25.         }
26.         // ... code ...
27. }
```

Código 18: Teste unitário com *xUnit* e *Moq*

Além dos repositórios, foram ainda simulados os serviços *HTTP* disponibilizados pela *Confluent Platform*, permitindo assim que não fosse necessário a execução dos serviços originais durante os testes realizados. Um exemplo do referido é a listagem dos *brokers* do *cluster* do *Kafka*, no qual é simulado o *URL*, o *endpoint* e o retorno do pedido *HTTP*.

```
1.  [Fact]
2.  public void GetTest()
3.  {
4.      // Setup
5.      // Mock HTTP Server
6.      var stubHttp = HttpMockRepository.At(API_URL_MOCK);
7.      stubHttp.Stub(x => x.Get("/brokers"))
8.          .Return("{\r\n  \"brokers\": [1, 2, 3]\r\n}")
9.          .OK();
10.
11.     // Act
12.     var result = controller.Get().Result as ObjectResult;
13.     var items = Assert.IsType<JArray>(result.Value);
14.
15.     // Assert
16.     Assert.Equal(3, items.Count);
17. }
```

Código 19: Teste unitário com *xUnit* e *HTTP Mock*

O resultado dos testes unitários para todas as classes do *Access System* encontra-se na Figura 40 em anexo.

6.3 *Web App*

A *Web App* foi implementada tal como o *design* apresentado na arquitetura, sendo assim desenvolvida a partir de um modelo base em *.NET* (C#) disponibilizado pela empresa. Inicialmente, juntamente com o meu colega Diogo Vigo estruturou-se a interface gráfica da aplicação em *mockups* (Anexo 3) de acordo com a interface do modelo disponibilizado. Finalizado isto, foi necessário analisar as funcionalidades já implementadas no projeto modelo e corrigir alguns aspetos do sistema para dar início ao desenvolvimento.

O projeto modelo estava configurado com o protocolo de comunicação *HTTPS*, utilizava uma abordagem *code-first* através do *Entity Framework* e *MySQL* como base de dados. As funcionalidades de gestão de utilizadores e funções já estavam implementadas, sendo apenas necessário alterar algumas propriedades dessas entidades para se adequarem à solução a desenvolver. Todo o processo de *login* e *logout*, mecanismos de autenticação e autorização já estavam implementados, bem como toda a interface base da aplicação, incluindo o menu de navegação e a navegação entre páginas.

Sendo assim, em conjunto com o meu colega, implementou-se algumas funcionalidades internas do sistema (de acesso restrito aos utilizadores com função de *SuperAdmin*), como a gestão de empresas e *URL's*. Além disso o desenvolvimento da minha parte limitou-se à implementação de funcionalidades de gestão de instalações e tópicos, e uma *dashboard* para a leitura de mensagens registadas nos tópicos, sendo que o desenvolvimento das restantes funcionalidades não foram da minha responsabilidade.

Para facilitar o desenvolvimento da aplicação, tanto na lógica de negócio como na interface gráfica, forma utilizadas as *frameworks* seguintes:

- ***.NET*** – *Framework* para desenvolvimento de aplicações;
- ***Entity Framework*** - Mapeamento das entidades armazenadas na base de dados *MySQL* através da abordagem *code-first*;
- ***Newtonsoft JSON*** – *Framework* utilizado para mapeamento de informação em *JSON*;
- ***RestSharp*** – Cliente utilizado para todos os pedidos *HTTP REST* realizados;
- ***JQuery*** – Biblioteca que permite simplificar os *scripts* de *JS* (*JavaScript*);
- ***Bootstrap*** – *Framework* para desenvolvimento de componentes de interface gráfica usando *JS*, *HTML* e *CSS*.

O modelo base disponibilizado pela empresa utilizava ainda uma classe modelo (*Audit*) para fins de auditoria, sendo que todas as classes modelos foram criadas através do conceito de herança sobre essa mesma classe, permitindo assim registar a data/hora e o utilizador que criou e que realizou a última edição sobre determinada entidade.

Desta forma, e tal como é demonstrado no Anexo 4, foi possível apresentar de forma visual o funcionamento da solução desenvolvida, permitindo assim a criação de tópicos, o registo de mensagens nos tópicos quando algum alerta é despoletado e a leitura desses registos através de vários parâmetros de filtragem.

7 Experimentação e Avaliação da Solução

A fase de experimentação e avaliação da solução desenvolvida foi realizada sobre o Sistema RDER e a *Web App* separadamente, uma vez que requerem abordagens de avaliação distintas. Desta forma, são definidas inicialmente as métricas de avaliação, após isso são apresentados os vários casos de teste e as metodologias de avaliação e por fim os resultados para o processo realizado.

7.1 Métricas

As métricas são os parâmetros utilizados na avaliação de ambos os sistemas, no entanto foram definidos separadamente para cada um dos sistemas, visto que cada sistema deve ser avaliado de forma particular.

7.1.1 Sistema RDER

Com base nos resultados esperados para a solução, o Sistema RDER deve permitir o armazenamento seguro de dados sensíveis, sendo que o conceito de segurança é definido pela encriptação e imutabilidade dos dados e, a disponibilidade do sistema. Em conjunto com isso deve possuir características de desempenho razoáveis para não comprometer o registo de dados, sendo que também devem ser considerados os custos de execução em ambiente *cloud* caso exista a necessidade de migrar o mesmo para qualquer plataforma *cloud*.

Uma vez que o conceito de imutabilidade é garantido pela arquitetura base do *Kafka* e não foi desenvolvido qualquer mecanismo de edição ou eliminação de dados, não foi tido em conta na avaliação do sistema, considerando assim este requisito como testado e validado. Assim, as métricas definidas para o Sistema RDER são as seguintes:

- **Desempenho dos mecanismos de encriptação** – Velocidade dos mecanismos de encriptação e desencriptação de dados;

- **Disponibilidade** – Garantir que o sistema se mantêm disponível e estável em caso de falha de nós do *cluster* do *Kafka*;
- **Desempenho** – Número de registos por segundo (*throughput*) suportadas pelo sistema;
- **Custo** – Custos de execução do sistema em ambiente *cloud*.

7.1.2 *Web App*

A *Web App* deve apresentar as funcionalidades já expostas, apresentar um nível de satisfação e usabilidade razoável e compatibilidade com dispositivos móveis. Dessa forma, as grandezas utilizadas para avaliar a *Web App* são:

- **Funcionalidade** – Verificação das funcionalidades implementadas;
- **Compatibilidade** – Verificação da compatibilidade com dispositivos móveis;
- **Satisfação e usabilidade** – Nível de satisfação e usabilidade apresentado.

7.2 Casos de teste e metodologia de avaliação

No caso do sistema RDER definiu-se os casos de testes para cada métrica selecionada, sendo posteriormente possível comparar os resultados dos vários testes para determinada métrica. Na *Web App*, por outro lado, apenas foi necessário definir um teste por métrica.

7.2.1 Sistema RDER

A metodologia de avaliação do Sistema RDER passa pela simulação de vários casos de teste, no qual são utilizadas máquinas virtuais com características de *hardware* idênticas às disponibilizadas pelo serviço de computação da plataforma *cloud AWS* (Instâncias do *Amazon EC2* [85]), e uma máquina local, apresentadas na Tabela 19. A utilização de máquinas virtuais permite assim a execução de testes variados e a obtenção dos custos de implementação em ambiente *cloud*.

Tabela 19: Máquinas para avaliação

Modelo	vCPU	Memória Ram (GiB)	Preço/Hora no *AWS*
T2.Small	1	2	0,0230 $
T2.Medium	2	4	0,0464 $
T2.Large	2	8	0,0928 $
Local	2	8	nd

Os casos de teste do Sistema RDER são definidos conforme as métricas referidas anteriormente, sendo executados 10 testes para cada caso, de forma a obter um conjunto de dados razoável para análise. Nos testes de desempenho de registo de mensagens utilizou-se a ferramenta *Apache JMeter* para medir o tempo de execução de cada operação, além disso, para cada teste

são indicadas as máquinas utilizadas na execução de cada componente (*Access System*, *Kafka Node* e *Confluent System*), apresentando desta forma a estrutura seguinte:

- **Desempenho dos mecanismos de encriptação:**
  - Teste da velocidade de encriptação e desencriptação de uma *string* de 512B, numa máquina local com o *Access System*;
  - Teste da velocidade de encriptação e desencriptação de uma *string* de 1kB, numa máquina local com o *Access System*;
  - Teste da velocidade de encriptação e desencriptação de uma *string* de 10kB, numa máquina local com o *Access System*.
- **Desempenho**:
  - *Throughput* de registo de mensagens num *cluster* de apenas um *Kafka Broker* diretamente para o *Kafka* e sem encriptação, numa máquina T2.Small com o *Kafka Node* e *Confluent System*;
  - *Throughput* de registo de mensagens num *cluster* de apenas um *Kafka Broker* através da *REST API* do *Access System* e sem encriptação, numa máquina T2.Small com o *Kafka Node* e *Confluent System* e, numa máquina local com o *Access System*;
  - *Throughput* de registo de mensagens num *cluster* de apenas um *Kafka Broker* através da *REST API* do *Access System* e com encriptação, numa máquina T2.Small com o *Kafka Node* e *Confluent System* e, numa máquina local com o *Access System*.
- **Custo:**
  - Relação custo/desempenho para um *cluster* de apenas um *Kafka Broker* numa máquina virtual T2.Small com o *Kafka Node* e *Confluent System*;
  - Relação custo/desempenho para um *cluster* de apenas um *Kafka Broker* numa máquina virtual T2.Medium com o *Kafka Node* e *Confluent System*;
  - Relação custo/desempenho para um *cluster* de apenas um *Kafka Broker* numa máquina virtual T2.Large com o *Kafka Node* e *Confluent System*.
- **Disponibilidade:**
  - Teste de disponibilidade num *cluster* de três *Kafka Brokers* no caso de um nó ficar indisponível, num *cluster* de três máquinas T2.Small com *Kafka Node*;
  - Teste de disponibilidade num *cluster* de três *Kafka Brokers* no caso de dois nós ficarem indisponíveis, num *cluster* de três máquinas T2.Small com *Kafka Node*.

Assim, de acordo com os vários cenários de avaliação apresentados para cada métrica, e dada a variedade de máquinas utilizadas, é possível:

- Verificar o impacto do tamanho da mensagem no tempo de encriptação e desencriptação da mesma;
- Analisar o impacto no desempenho causado pelo componente de acesso desenvolvido *Access System* e pelo mecanismo de encriptação;

- Calcular a relação custo/desempenho para máquinas virtuais idênticas às apresentadas pelo serviço de computação da *AWS*. O cálculo é efetuado através do valor de *throughput* e do custo por hora para cada tipo de máquina disponibilizada pelo serviço *cloud*. Além disso, permite identificar o impacto no desempenho do sistema, conforme o *hardware* utilizado;
- Verificar se o sistema se comporta como é espectável quando algum nó do *cluster* fica indisponível, permitindo assim testar o mecanismo de tolerância a falhas apresentado pelo *Zookeeper*. É o único caso em que é necessário testar sobre um *cluster* com mais do que um *Kafka Broker*.

7.2.2 *Web App*

Apenas foi possível testar as métricas de funcionalidade, compatibilidade e usabilidade da *Web App* após a mesma ser utilizada de forma detalhada. Desta forma o caso de teste da *Web App* passa pela utilização intensiva da mesma por parte das pessoas que a vão utilizar ou das que a desenvolveram.

Assim, a metodologia de avaliação escolhida para a métrica de satisfação e usabilidade foi a realização de um inquérito de satisfação e usabilidade em conjunto com o meu colega Diogo Vigo através do *Google Forms*, tal como é apresentado no Anexo 5. O inquérito foi preenchido por pessoas externas à empresa e desenvolvido com um guião base (Figura 57) de forma a facilitar todo o processo, sendo assim, as questões definidas foram:

- Q1 - Considerou o sistema suficientemente intuitivo para seguir o guião de forma eficiente?
- Q2 - Considerou a navegação entre páginas e operações prática e intuitiva?
- Q3 - Considerou o texto apresentado adequado e percetível?
- Q4 - Considerou o tipo/tamanho de letra legível?
- Q5 - Considerou o esquema de cores adequado?
- Q6 - Considerou o tempo de resposta das operações tolerável?
- Q7 - Em caso de uso estaria satisfeito com o sistema?
- Q8 - Durante o guião alguma das operações não correu da forma esperada?
    - Q9 - Se sim, quais?
- Q10 - Quais a(s) operação(s) mais intuitivas?
- Q11 - Quais a(s) operação(s) menos intuitivas?

A Tabela 20 representa a escala utilizada para resposta às questões do inquérito, com base na escala de *Likert* [86].

Tabela 20: Escala de respostas ao inquérito

Escala	Descrição
1	Muito insatisfeito
2	Parcialmente insatisfeito
3	Nem satisfeito nem insatisfeito
4	Parcialmente satisfeito
5	Muito satisfeito

Para as métricas de funcionalidades e compatibilidade, são realizados testes funcionais de modo a verificar manualmente quais as funcionalidades implementadas relativamente às identificadas na fase inicial e se existe compatibilidade com dispositivos móveis.

7.3 Resultados

Após a execução de todos os casos de teste relativos ao sistema RDER e da análise da *Web App* e preenchimento do inquérito sobre a mesma, foi possível analisar todos os resultados e avaliar a solução desenvolvida.

7.3.1 Sistema RDER

Todos os testes desenvolvidos para o sistema RDER foram executados na mesma máquina, incluindo os testes que envolveram a utilização de máquinas virtuais. Sendo assim, não deve ser considerado qualquer impacto negativo causado pela rede, como aconteceria num cenário real, sendo que o único fator prejudicial que pode existir é a ligação entre a máquina local e as máquinas virtuais executadas na máquina local.

Os testes realizados ao mecanismo de encriptação forma executados sobre o algoritmo *AES* no modo *CTR*, utilizando uma chave de 128 bits, tal como desenvolvido. Os resultados obtidos são apresentados na Tabela 21.

Tabela 21: Teste de desempenho da encriptação e desencriptação

	Tempo de encriptação (ms)			Tempo de desencriptação (ms)		
Teste	**512B**	**1kB**	**10kB**	**512B**	**1kB**	**10kB**
1	0,4616	0,4079	0,7947	0,6225	0,6706	1,4928
2	0,6346	0,6638	0,7494	0,6571	0,4380	1,4034
3	0,4646	0,4115	0,6347	0,3935	0,4049	0,6745
4	0,6085	0,6148	0,9087	0,5997	0,4453	1,3921
5	0,6285	0,6561	1,0664	0,6138	0,4474	1,3123
6	0,4272	0,4095	0,7501	0,4690	0,4147	0,9992
7	0,4430	0,4259	0,6236	0,3970	0,5809	0,8582
8	0,5352	0,5425	1,0745	0,7074	0,7286	1,0286
9	0,4479	0,4426	0,6662	0,4468	0,4136	1,0351
10	0,4350	0,4336	0,6941	0,4145	0,4035	0,8782
Total (ms)	0,5086	0,5008	0,7962	0,5321	0,4948	1,1074

O gráfico exposto na Figura 32 baseado nos dados da tabela anterior apresenta tempos de execução bastante idênticos para mensagens com tamanho de 512B e 1kB, além disso, os tempos de execução para mensagens com tamanho de 10kB não apresentam tanta discrepância relativamente aos restantes, apesar do tamanho da mensagem ser 10 vezes superior. Assim, o tamanho da mensagem não apresenta tanto impacto no tempo de encriptação da mesma.

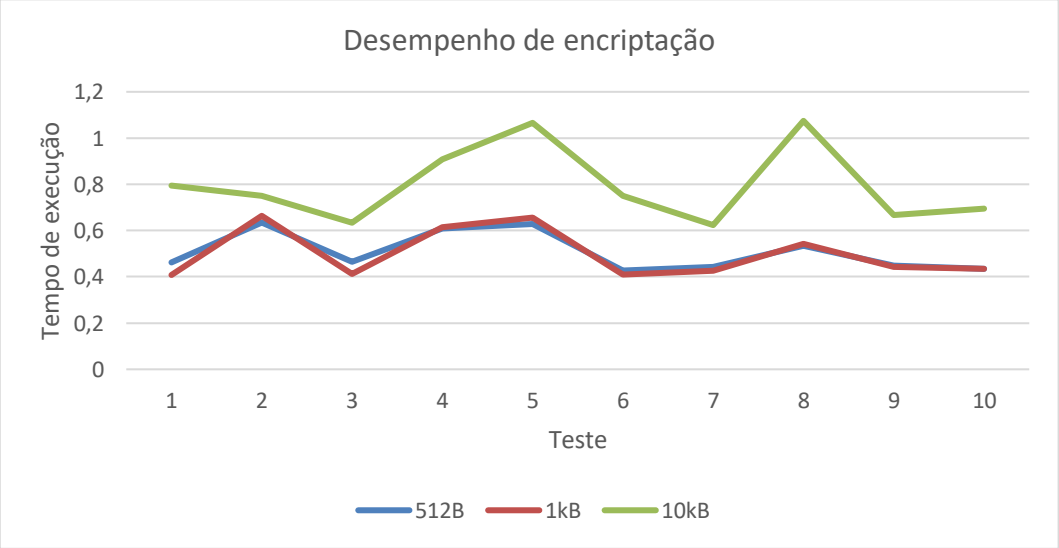


Figura 32: Gráfico do desempenho de encriptação

Tal como no processo de encriptação, o teste de desencriptação representado pela Figura 33 apresenta resultados bastante idênticos, apesar do tempo de execução para mensagens com tamanho de 512B e 1kB diferirem mais entre si comparativamente ao processo de encriptação. Sendo assim, e tal como no gráfico anterior, mensagens com tamanho superior não afetam de forma considerável o tempo de execução.

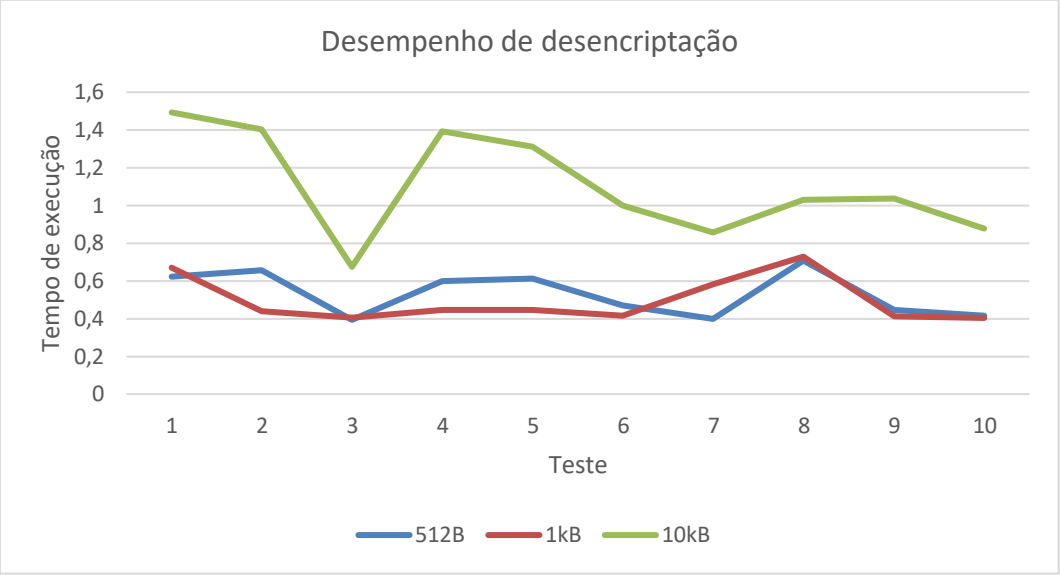


Figura 33: Gráfico do desempenho de desencriptação

Relativamente à comparação entre os processos de encriptação e desencriptação, presente na Figura 34, os resultados apresentados para as mensagens com tamanho de 512B e 1kB são praticamente idênticos, no entanto para mensagens com tamanho de 10kB é notável uma diferença significativa, no qual o processo de desencriptação apresenta um tempo de execução de cerca de 39% mais demorado.

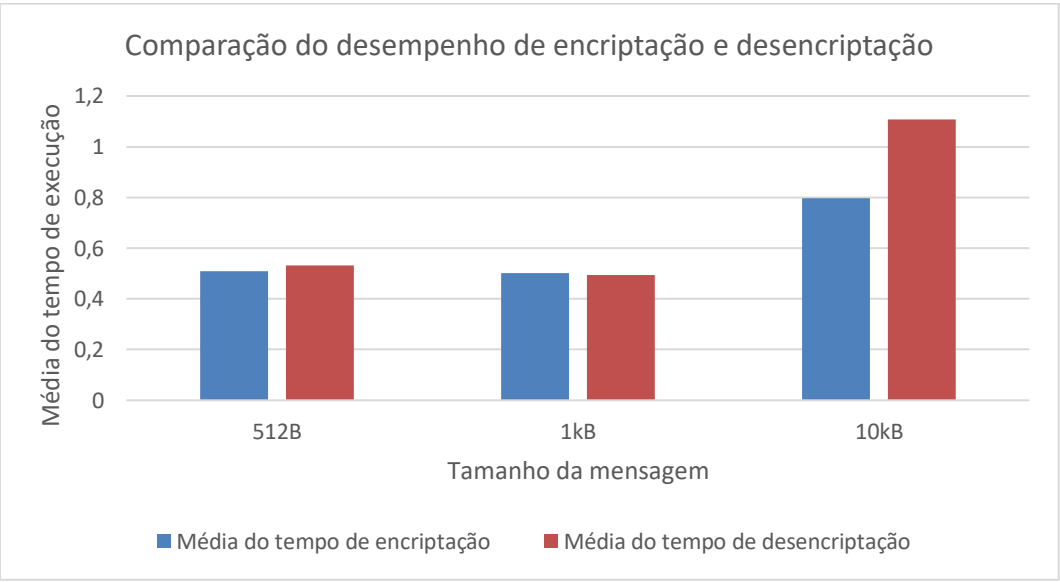


Figura 34: Gráfico de comparação do desempenho de encriptação e desencriptação

Para a avaliação de desempenho, utilizou-se três casos de teste, sendo que cada um foi implementado com componentes e tarefas distintas, de forma a perceber o impacto desses mesmos componentes e tarefas no desempenho de registo de mensagens. Assim, para os resultados apresentados na Tabela 22 relativos ao registo de 10000 mensagens com tamanho de 512B, os casos de teste forma os seguintes:

- **Caso de teste 1 (CT1)** – Registo diretamente sobre o *Kafka*;
- **Caso de teste 2 (CT2)** – Registo sem encriptação através do *Access System*;
- **Caso de teste 3 (CT3)** – Registo com encriptação através do *Access System*.

Tabela 22: Teste de desempenho de registo de mensagens

	Tempo de registo de 10000 mensagens com 512B (ms)		
Teste	**CT1**	**CT2**	**CT3**
1	1936	3485	8426
2	1889	2740	6710
3	1765	2529	6308
4	1722	2615	6626
5	1602	2238	6367
6	1666	2149	6708
7	1364	2340	6684
8	1267	2701	7041
9	1248	2037	7188
10	1439	1822	6775
***Throughput* (msgs/sec)**	6290	4056	1453

De acordo com o gráfico apresentado na Figura 35 e com os resultados da tabela anterior, é demonstrado que o tempo de execução do CT2 é ligeiramente superior ao CT1, revelando um impacto de 55% no desempenho causado pelo *Access System*. Contudo, é sobretudo notável a discrepância no tempo de execução do CT3 relativamente aos outros casos de teste, no qual o mecanismo de encriptação adicional relativamente ao CT2 afeta o desempenho em cerca de 179%, sendo que no CT2 é obtida uma média de *throughput* de 4056 mensagens por segundo enquanto que no CT3 é apenas obtido um valor médio de 1453 registos por segundo.

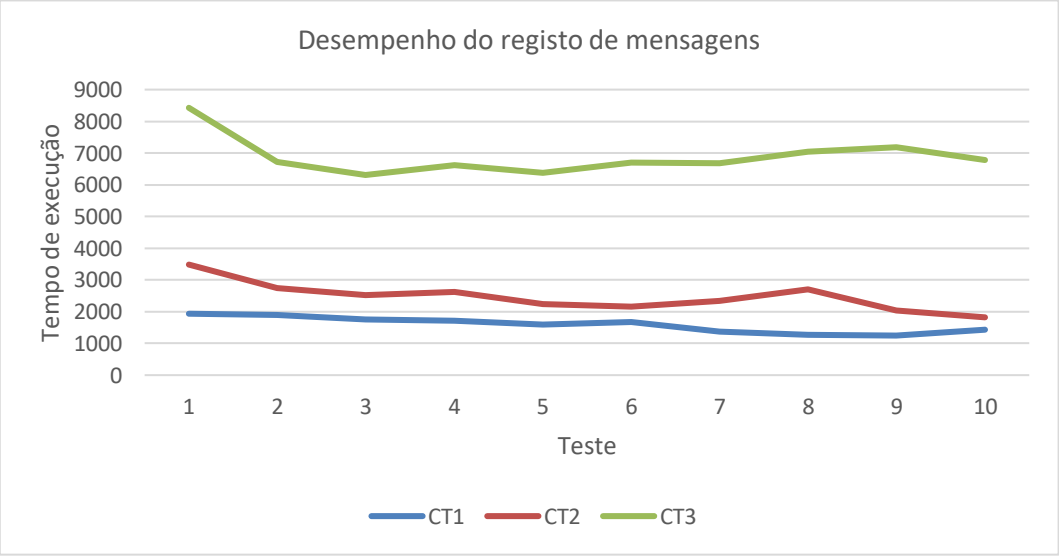


Figura 35: Gráfico de desempenho de registo de mensagens

A avaliação do custo/desempenho foi obtida após o teste do tempo de registo de 10000 mensagens de 512B, em máquinas virtuais idênticas às disponibilizadas na plataforma *cloud AWS*, sendo que os resultados obtidos são apresentados na Tabela 23.

Tabela 23: Avaliação do custo/desempenho em ambiente *cloud* para diferentes máquinas

	Tempo de registo de 10k mensagens com 512B (ms)		
Teste	**T2.Small**	**T2.Medium**	**T2.Large**
1	1936	1168	374
2	1889	798	330
3	1765	1139	236
4	1722	653	274
5	1602	696	271
6	1666	692	223
7	1364	631	180
8	1267	619	209
9	1248	925	220
10	1439	678	170
***Throughput* (msgs/sec)**	6290	12502	40209
Custo por hora ($)	0,0230	0,0464	0,0928
Custo/Desempenho	$3{,}65 \times 10^{-6}$	$3{,}7 \times 10^{-6}$	2.3×10^{-6}

Uma vez que os testes de custo/desempenho foram executados em máquinas com *hardware* diferente, foi possível analisar o impacto no desempenho causado pelo *hardware* utilizado. Assim, e como é apresentado no gráfico da Figura 36, é notória uma diferença significativa de desempenho causada pela alteração de *hardware*, sendo que a máquina *T2.Medium* apresenta um desempenho cerca de 99% superior à máquina *T2.Small*, e a máquina *T2.Large* apresenta um desempenho cerca de 222% superior à máquina *T2. Medium*.

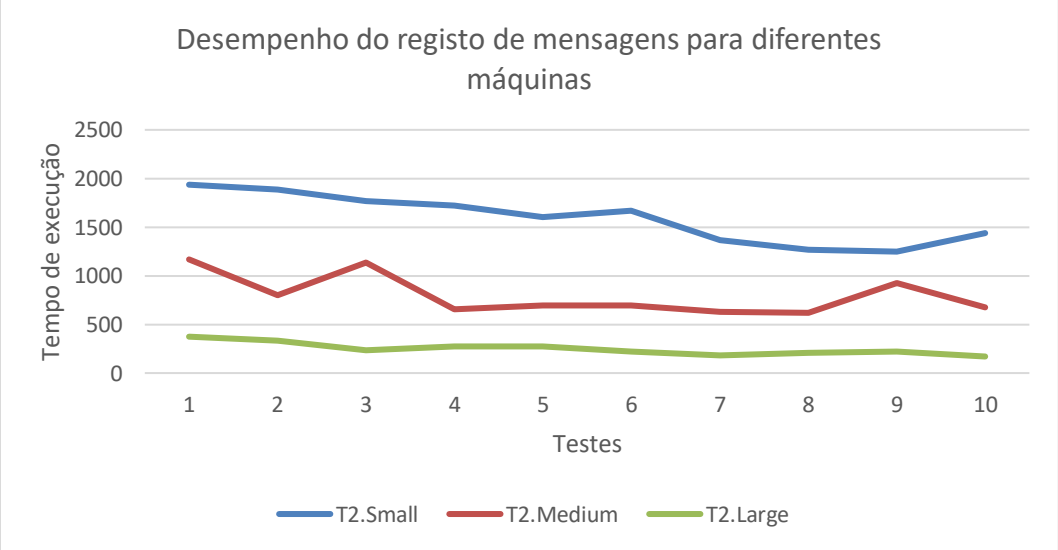


Figura 36: Gráfico de desempenho de registo de mensagens para diferentes máquinas

Assim, de acordo com o desempenho de cada máquina e o respetivo custo por hora, é possível afirmar que a utilização de uma máquina *T2.Large* garante o melhor custo por desempenho, sendo que às máquinas *T2.Small* e *T2.Medium* apresentam um custo por desempenho idêntico.

Por fim, o teste de disponibilidade do *Cluster* do *Kafka*, realizado sobre um *cluster* de três máquinas *T2.Small* com o componente *Kafka Node*, consistiu em tornar indisponível máquina após máquina e verificar de que forma isso afetava o *cluster*. Assim, após a primeira máquina ficar indisponível, o *cluster* manteve o funcionamento com apenas dois nós, e após a segunda máquina ficar indisponível, o sistema também se manteve disponível. É assim possível garantir que o mecanismo de tolerância a falhas do *Zookeeper* funciona como esperado, assegurando a disponibilidade do sistema no caso de falha de dois nós num *cluster* de três nós.

7.3.2 *Web App*

Após o desenvolvimento da *Web App* foi possível verificar que todos os casos de uso, apresentados anteriormente na Figura 22, foram implementados, garantindo assim o funcionamento de que todas as funcionalidades inicialmente definidas.

A aplicação foi desenvolvida com uma interface gráfica responsiva, permitindo a compatibilidade da mesma com dispositivos móveis, o que garante um nível de versatilidade considerável, tal como é apresentado na Figura 37.

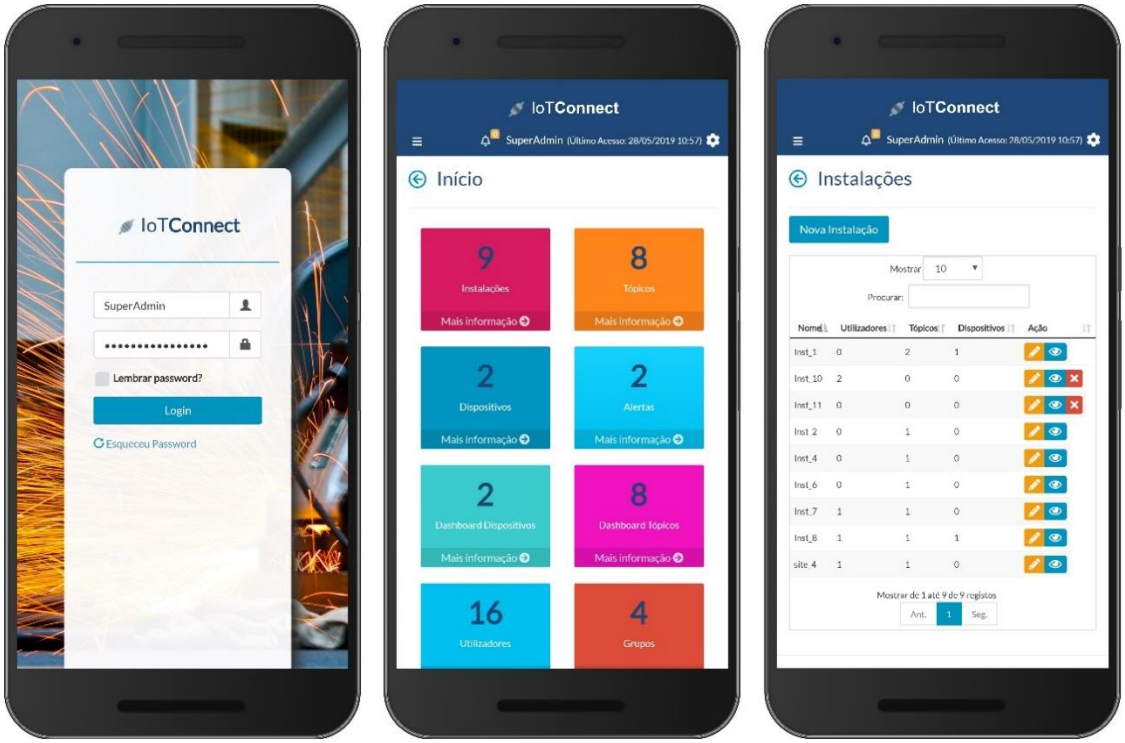


Figura 37: *Web App* – Compatibilidade com dispositivos móveis

Relativamente ao inquérito de satisfação e usabilidade criado e realizado por 16 pessoas, foi possível obter resultados de satisfação e usabilidade da *Web App* desenvolvida, sendo que os mesmos são apresentados na Tabela 24 para as questões de 1 (Q1) a 7 (Q7).

Tabela 24: Resultados do inquérito para as questões Q1 a Q7

Questão	1	2	3	4	5	Média
Q1	0%	0%	13%	50%	38%	4,3
Q2	0%	0%	0%	56%	44%	4,4
Q3	0%	0%	13%	50%	38%	4,3
Q4	0%	0%	0%	50%	50%	4,5
Q5	0%	0%	0%	19%	81%	4,8
Q6	0%	0%	19%	75%	6%	3,9
Q7	0%	0%	6%	56%	38%	4,3

Assim, conforme os resultados representados pela tabela anterior, conclui-se que a *Web App* apresenta um nível de usabilidade e satisfação bastante razoável, uma vez que a média da maioria das questões é superior a quatro (Parcialmente satisfeito na escala de *Likert*), com um valor médio global de 4,3. A questão 7 (Q7), no qual é questionado o nível de satisfação geral no caso de utilização da aplicação, teve um resultado médio de 4,3, exatamente igual ao valor médio de todas as questões, o que apresenta um elevado nível de consistência nas respostas.

No entanto, é visível um nível de satisfação inferior na Q6 comparativamente com as restantes questões, sendo a mesma referente aos tempos de resposta das operações. Esta disparidade de satisfação pode ser causada pela falta de otimização a nível de interface gráfica ou associada diretamente aos serviços disponibilizados pelo Sistema RDER.

Além disso, nenhum dos participantes identificou qualquer operação que não tenha sido executada conforme o expectável (Q8 e Q9).

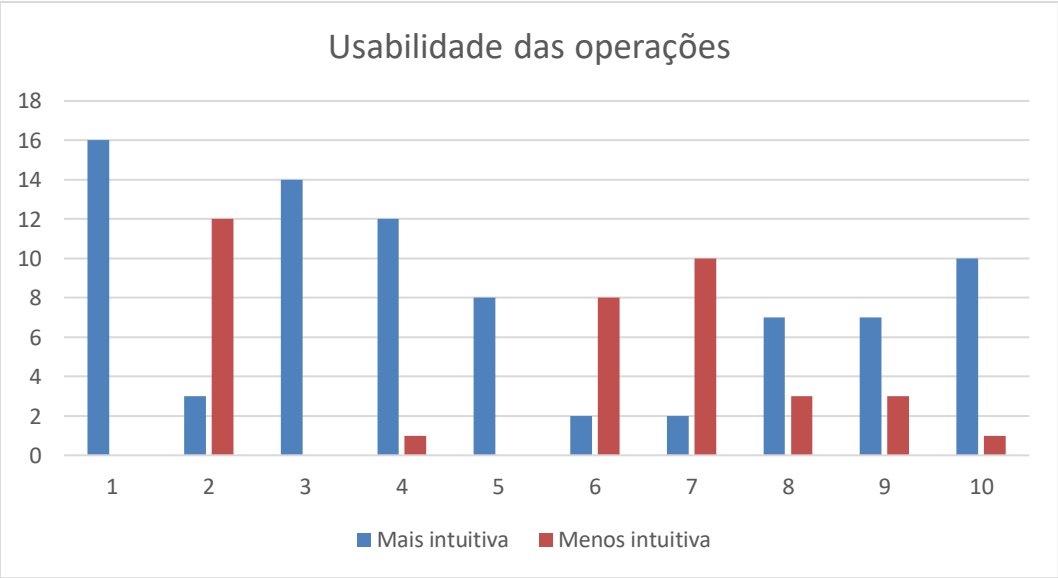


Figura 38: Usabilidade das operações do guião

A Figura 38 apresenta os resultados referentes à usabilidade das operações presentes no guião, obtidos através das questões 10 (Q10) e 11 (Q11), no qual as operações nº 1, 3, 4 e 10 estão presentes em mais de metade dos inquéritos como sendo as operações mais intuitivas, sendo que a operação nº 1 (*Login* na *Web App*) está presente em exatamente todos os inquéritos.

Das operações identificadas como menos intuitivas, são destacadas as operações nº 2, 6 e 7, sendo que a operação nº 2 (Alterar *password*) pode ter sido indicada devido à sua acessibilidade e visibilidade, uma vez que é apenas acessível através de um botão no canto superior direito da aplicação. O baixo nível de usabilidade nas operações 6 e 7 pode ser causado pelo facto das mesmas necessitarem de várias associações com outras entidades da aplicação, o que pode confundir o utilizador.

8 Conclusões

Apesar da consciencialização para o tema da STT e das crescentes ações de prevenção de acidentes de trabalho, continuam a ocorrer inúmeros acidentes de trabalho, associado a isso existe ainda a necessidade de fornecer dados fiáveis às empresas prestadoras de serviços de seguros de forma a confirmar a ocorrência do sinistro.

Assim, o presente problema baseia-se em garantir que a informação fornecida às empresas referidas é fiável e permite assegurar a veracidade de determinado sinistro, garantindo que não se trata de qualquer ato fraudulento.

A abordagem inicialmente proposta pela empresa foi idealizada sobre tecnologias *Blockchain*, uma vez que apresentavam características de armazenamento imutável e seguro de informação, o que permitia armazenar dados críticos possivelmente associados a acidentes de trabalho. Deste modo, grande parte da pesquisa foi realizada sobre os conceitos fundamentais do *Blockchain* e das tecnologias associadas ao mesmo, contudo foram de igual forma pesquisadas e avaliadas outras tecnologias com características semelhantes que similarmente poderiam solucionar o problema em questão, o que incluiu tecnologias de bases de dados e a tecnologia *Apache Kafka*.

Após todo processo de pesquisa e análise, foi rejeitada a utilização de tecnologias *Blockchain* uma vez que a sua utilização não era adequada para a solução idealizada, sendo que a decisão da tecnologia final recaiu principalmente sobre o *Apache Kafka* com a utilização da *Confluent Platform* e com um mecanismo de encriptação baseado no algoritmo *AES*, apresentando-se como a solução mais completa e equilibrada para os requisitos definidos.

8.1 Objetivos alcançados

A resposta ao problema apresentado e simultaneamente à questão principal deste documento, “Como comprovar e facilitar todo o processo pós acidente?”, foi conseguida através do desenvolvimento de uma prova de conceito baseada no *Apache Kafka* e na *Confluent Platform*,

acessível através de uma *REST API* desenvolvida em *.Net Core* e visualizada através de uma *Web App* desenvolvida essencialmente em *.Net*.

Isto permitiu construir uma solução para armazenamento imutável de dados críticos, dada a arquitetura do *Kafka*, e num formato ilegível, uma vez que a informação é encriptada pelo algoritmo de encriptação *AES* no modo *CTR*, através da utilização de chaves de 128 bits. Além disto, a solução apresenta ainda medidas de segurança adicionais, como mecanismos de tolerância a falhas e a implementação do protocolo *HTTPS* em determinados componentes. A *Web App* permitiu a integração do sistema RDER com o sistema *IoT* desenvolvido pelo meu colega, garantindo todo o processo de registo e leitura de dados críticos captados pelo sistema *IoT* e a visualização gráfica de todos os acontecimentos.

Deste modo, o objetivo fundamental do projeto desenvolvido e documentado no presente documento foi alcançado, uma vez que foi possível o registo e leitura de dados críticos num sistema de armazenamento seguro e imutável. Contudo, tal como é apresentado na arquitetura da prova de conceito existem medidas de segurança não implementadas comparativamente à arquitetura definida como a mais adequada, tal como a implementação do protocolo *HTTPS* nos componentes relativos ao *Kafka*.

Os testes de avaliação executados para o Sistema RDER permitiram concluir que o desempenho do mecanismo de encriptação é apenas ligeiramente afetado pelo tamanho da mensagem a encriptar, porém apresenta um enorme impacto no desempenho do sistema, de cerca de 179%. Foi possível também concluir que o *hardware* da máquina é relevante para o desempenho do sistema e que a implementação mais indicada relativamente ao custo/desempenho do mesmo em ambiente *cloud* (plataforma *AWS*) é através da utilização de máquinas *T2.Large*. Por fim, os testes permitiram também garantir o funcionamento do mecanismo de tolerância a falhas do *Zookeeper*, assegurando assim a disponibilidade do sistema.

Os testes realizados para a *Web App* permitiram garantir o funcionamento de todas as funcionalidades inicialmente definidas e o funcionalmente da mesma em dispositivos móveis através da sua interface gráfica responsiva. Além disso, com base no inquérito de satisfação e usabilidade foi possível obter um nível de aceitação da *Web App* bastante satisfatório, no qual o valor médio das respostas foi de 4,3 (baseado na escala de *Likert* de 1 a 5), além disso foi ainda possível compreender quais os aspetos a melhorar na aplicação.

8.2 Limitações e trabalho futuro

A principal limitação foi a impossibilidade de desenvolver a solução de acordo com a arquitetura definida, uma vez que o número de servidores que a solução requeria não foi acessível, impedindo assim a execução da solução num ambiente ideal e que garantia mais medidas de segurança.

A limitação referida apresenta-se também como trabalho futuro, uma vez que a arquitetura definida como ideal requer a execução dos componentes em vários servidores, ao contrário da

prova de conceito, permitindo assim medidas de segurança adicionais. A implementação da solução em várias máquinas possibilitaria ainda verificar o impacto causado pela rede no desempenho do sistema, através da execução de testes em condições idênticas aos realizados e da comparação dos mesmos com os testes apresentados neste documento. Também em falta na prova de conceito está a implementação do protocolo *HTTPS* para todos os componentes que permitam a comunicação, o que garantiria um nível de segurança elevado na comunicação entre componentes.

Após a avaliação da solução através dos testes realizados, é sobretudo relevante o impacto causado pelo mecanismo de encriptação no desempenho do sistema, sendo que a otimização do algoritmo utilizado seria um trabalho futuro bastante significativo para a solução. Associado ao mecanismo de encriptação, seria ainda vantajoso a utilização de um serviço *cloud* para armazenamento das chaves de 128 bits utilizadas, que apesar de apresentar custos, garante um nível de segurança superior ao mecanismo atual e permite a migração do sistema independentemente da máquina no qual é executado.

A avaliação da *Web App* através do inquérito realizado possibilitou a definição de melhorias relativamente à mesma, sendo que seria vantajoso otimizar a interface gráfica e simultaneamente os serviços utilizados pela aplicação, de maneira a reduzir o tempo de resposta das operações. Além disso, seria também favorável aperfeiçoar a interface gráfica de forma a simplificar o acesso à operação 2 do guião (Alterar *password*), bem como facilitar e agilizar as operações 6 (Criar dispositivo) e 7 (Criar alerta).

Referências

[1] «Safety and health at work». [Em linha]. Disponível em: http://www.ilo.org/global/topics/safety-and-health-at-work/lang--de/index.htm. [Acedido: 24-Out-2018].

[2] admin, «InCloud for Safemed, o futuro para a segurança e saúde no trabalho», *Ábaco Consultores*, 11-Nov-2017. .

[3] «Accidents at work statistics - Statistics Explained». [Em linha]. Disponível em: https://ec.europa.eu/eurostat/statistics-explained/index.php/Accidents_at_work_statistics#Accidents_2010_to_2015_-_absolute_changes. [Acedido: 24-Out-2018].

[4] «Gabinete de Estratégia e Planeamento (GEP/MTSSS) - Estatística». [Em linha]. Disponível em: http://www.gep.msess.gov.pt/estatistica/acidentes/. [Acedido: 27-Out-2018].

[5] «Lei 98/2009, 2009-09-04», *Diário da República Eletrónico*. [Em linha]. Disponível em: https://dre.pt. [Acedido: 27-Out-2018].

[6] «Constituição da República Portuguesa». [Em linha]. Disponível em: http://www.parlamento.pt/Legislacao/Paginas/ConstituicaoRepublicaPortuguesa.aspx. [Acedido: 24-Out-2018].

[7] «Lei 102/2009, 2009-09-10», *Diário da República Eletrónico*. [Em linha]. Disponível em: https://dre.pt. [Acedido: 27-Out-2018].

[8] «Comunicações e Autorizações Obrigatórias». [Em linha]. Disponível em: http://www.act.gov.pt/(pt-PT)/CentroInformacao/ComunicacoeseAutorizacoesObrigatorias/Paginas/default.aspx. [Acedido: 27-Out-2018].

[9] «Acidentes de Trabalho por negligência da empresa», *RP Associados*. .

[10] P. Koen *et al.*, «Providing Clarity and A Common Language to the "Fuzzy Front End"», *Research-Technology Management*, vol. 44, n. 2, pp. 46–55, Mar. 2001.

[11] G. Lancaster, «Implementing value strategy through the value chain», 2000.

[12] W. Ulaga e A. Eggert, «Relationship Value and Relationship Quality: Broadening the Nomological Network of Business-to-Business Relationships», Mar. 2006.

[13] T. Woodall, «Conceptualising "Value for the Customer": An Attributional, Structural and Dispositional Analysis», p. 45, 2003.

[14] S. Nakamoto, «Bitcoin: A Peer-to-Peer Electronic Cash System», p. 9, 2008.

[15] «Distributed Ledger Technology: beyond block chain», p. 88, 2016.

[16] «Blockchains & Distributed Ledger Technologies», *BlockchainHub*. .

[17] S. Seth, «Consensus Mechanism (Cryptocurrency)», *Investopedia*, 03-Abr-2018. [Em linha]. Disponível em: https://www.investopedia.com/terms/c/consensus-mechanism-cryptocurrency.asp. [Acedido: 29-Out-2018].

[18] O. Momoh, «Proof of Stake (PoS)», *Investopedia*, 03-Mai-2017. [Em linha]. Disponível em: https://www.investopedia.com/terms/p/proof-stake-pos.asp. [Acedido: 29-Out-2018].

[19] «What is Delegated Proof of Stake?», *Lisk*. [Em linha]. Disponível em: https://lisk.io/academy/blockchain-basics/how-does-blockchain-work/delegated-proof-of-stake. [Acedido: 29-Out-2018].

[20] «What is Proof of Authority Consensus? (PoA) Staking Your Identity», *Blockonomi*, 05-Jul-2018. [Em linha]. Disponível em: https://blockonomi.com/proof-of-authority/. [Acedido: 29-Out-2018].

[21] S. Seth, «Proof of Elapsed Time (Cryptocurrency)», *Investopedia*, 04-Abr-2018. [Em linha]. Disponível em: https://www.investopedia.com/terms/p/proof-elapsed-time-cryptocurrency.asp. [Acedido: 29-Out-2018].

[22] M. Castro e B. Liskov, «Practical Byzantine Fault Tolerance», p. 14, 1999.

[23] S. Lee, «Explaining Directed Acylic Graph (DAG), The Real Blockchain 3.0», *Forbes*. [Em linha]. Disponível em: https://www.forbes.com/sites/shermanlee/2018/01/22/explaining-directed-acylic-graph-dag-the-real-blockchain-3-0/. [Acedido: 29-Out-2018].

[24] «Demistifying Blockchain: A Detailed Step-by-Step Explanation», *Reacle*. .

[25] «Developer Guide - Bitcoin». [Em linha]. Disponível em: https://bitcoin.org/en/developer-guide#block-chain. [Acedido: 29-Out-2018].

[26] «What Is Hashing? Under The Hood Of Blockchain», *Blockgeeks*. [Em linha]. Disponível em: https://blockgeeks.com/guides/what-is-hashing/. [Acedido: 29-Out-2018].

[27] «What is a Merkle Tree? Beginner's Guide to this Blockchain Component», *Blockonomi*, 09-Jul-2018. [Em linha]. Disponível em: https://blockonomi.com/merkle-tree/. [Acedido: 30-Out-2018].

[28] J. P. Kelleher, «Bitcoin Mining», *Investopedia*, 21-Abr-2014. [Em linha]. Disponível em: https://www.investopedia.com/terms/b/bitcoin-mining.asp. [Acedido: 30-Out-2018].

[29] «Mining - Bitcoin Wiki». [Em linha]. Disponível em: https://en.bitcoin.it/wiki/Mining. [Acedido: 30-Out-2018].

[30] «Proof of work - Bitcoin Wiki». [Em linha]. Disponível em: https://en.bitcoin.it/wiki/Proof_of_work. [Acedido: 30-Out-2018].

[31] «What are smart contracts on blockchain?», *Blockchain Unleashed: IBM Blockchain Blog*, 02-Jul-2018. [Em linha]. Disponível em: https://www.ibm.com/blogs/blockchain/2018/07/what-are-smart-contracts-on-blockchain/. [Acedido: 31-Out-2018].

[32] «Banking Is Only The Beginning: 42 Big Industries Blockchain Could Transform», *CB Insights Research*, 08-Ago-2018. [Em linha]. Disponível em: https://www.cbinsights.com/research/industries-disrupted-blockchain/. [Acedido: 31-Out-2018].

[33] *Bitcoin Core integration/staging tree*. Bitcoin, 2018.

[34] *Official Go implementation of the Ethereum protocol*. ethereum, 2018.

[35] *Hyperledger Fabric is a blockchain framework implementation and one of the Hyperledger projects hosted by The Linux Foundation*. Hyperledger, 2018.

[36] *Decentralized cryptocurrency blockchain daemon implementing the XRP Ledger in C++*. Ripple, 2018.

[37] *Meet BigchainDB. The blockchain database.* BigchainDB, 2018.

[38] *A permissioned implementation of Ethereum supporting data privacy*. J.P. Morgan, 2018.

[39] *Corda is an open source blockchain project, designed for business from the start. Only Corda allows you to build interoperable blockchain networks that transact in strict privacy.* Corda, 2018.

[40] *Stellar-core is the backbone of the Stellar network. It maintains a local copy of the ledger, communicating and staying in sync with other instances of stellar-core on the network.* stellar, 2018.

[41] *Source code for multichaind, multichain-cli and multichain-util*. MultiChain, 2018.

[42] *Openchain node reference implementation*. Openchain, 2018.

[43] «Newest “couchdb” Questions», *Stack Overflow*. [Em linha]. Disponível em: https://stackoverflow.com/questions/tagged/couchdb. [Acedido: 10-Jun-2019].

[44] «Newest “couchbase” Questions», *Stack Overflow*. [Em linha]. Disponível em: https://stackoverflow.com/questions/tagged/couchbase. [Acedido: 10-Jun-2019].

[45] «Newest “datomic” Questions», *Stack Overflow*. [Em linha]. Disponível em: https://stackoverflow.com/questions/tagged/datomic. [Acedido: 10-Jun-2019].

[46] «Overview — Apache CouchDB® 2.3 Documentation». [Em linha]. Disponível em: http://docs.couchdb.org/en/stable/index.html. [Acedido: 07-Jun-2019].

[47] «Couchbase Documentation». [Em linha]. Disponível em: https://developer.couchbase.com/documentation/server/3.x/admin/Couchbase-intro.html. [Acedido: 07-Jun-2019].

[48] «What Is Datomic Cloud? | Datomic». [Em linha]. Disponível em: https://docs.datomic.com/cloud/index.html. [Acedido: 07-Jun-2019].

[49] «Apache Kafka». [Em linha]. Disponível em: https://kafka.apache.org/. [Acedido: 05-Jan-2019].

[50] «It's Okay To Store Data In Kafka», *Confluent*, 15-Set-2017. [Em linha]. Disponível em: https://www.confluent.io/blog/okay-store-data-apache-kafka/. [Acedido: 07-Jan-2019].

[51] *Mirror of Apache Kafka. Contribute to apache/kafka development by creating an account on GitHub*. The Apache Software Foundation, 2019.

[52] «Apache Kafka Documentation», *Apache Kafka*. [Em linha]. Disponível em: https://kafka.apache.org/documentation/. [Acedido: 09-Jan-2019].

[53] «ZooKeeper Internals». [Em linha]. Disponível em: https://zookeeper.apache.org/doc/r3.4.13/zookeeperInternals.html. [Acedido: 05-Jan-2019].

[54] «Zookeeper Atomic Broadcast Protocol (ZAB) and implementation of Zookeeper. - CloudKarafka, Apache Kafka Message streaming as a Service». [Em linha]. Disponível em: https://www.cloudkarafka.com/blog/2018-07-04-cloudkarafka-zab.html. [Acedido: 06-Jun-2019].

[55] «What is Confluent Platform? — Confluent Platform». [Em linha]. Disponível em: https://docs.confluent.io/current/platform.html. [Acedido: 07-Jan-2019].

[56] «What is the difference between asymmetric & symmetric encryption? - Quora». [Em linha]. Disponível em: https://www.quora.com/What-is-the-difference-between-asymmetric-symmetric-encryption#!n=12. [Acedido: 07-Jan-2019].

[57] «O RGPD». [Em linha]. Disponível em: https://www.cnpd.pt/bin/rgpd/rgpd.htm. [Acedido: 26-Nov-2018].

[58] «O que são dados pessoais?» [Em linha]. Disponível em: https://ec.europa.eu/info/law/law-topic/data-protection/reform/what-personal-data_pt. [Acedido: 27-Nov-2018].

[59] B. D. Journal, «Here's how GDPR and the blockchain can coexist», *The Next Web*, 26-Jul-2018. [Em linha]. Disponível em:

https://thenextweb.com/syndication/2018/07/26/gdpr-blockchain-cryptocurrency/. [Acedido: 06-Jun-2019].

[60] V. Ferrari, «EU Blockchain Observatory and Forum Workshop on GDPR, Data Policy and Compliance», *SSRN Electronic Journal*, 2018.

[61] «Amazon QLDB», *Amazon Web Services, Inc.* [Em linha]. Disponível em: https://aws.amazon.com/qldb/. [Acedido: 14-Jan-2019].

[62] «Hyperledger Fabric», *Hyperledger*. .

[63] «BigchainDB - The blockchain database.», *BigchainDB*. [Em linha]. Disponível em: https://www.bigchaindb.com/. [Acedido: 06-Dez-2018].

[64] «Quorum | J.P. Morgan». [Em linha]. Disponível em: https://www.jpmorgan.com/global/Quorum. [Acedido: 06-Dez-2018].

[65] «Corda Platform», *r3.com*, 29-Jun-2018. .

[66] Q. Nasir, I. A. Qasse, M. Abu Talib, e A. B. Nassif, «Performance Analysis of Hyperledger Fabric Platforms», *Security and Communication Networks*, 2018. [Em linha]. Disponível em: https://www.hindawi.com/journals/scn/2018/3976093/. [Acedido: 06-Dez-2018].

[67] «Load testing - Attempt to achieve high throughput in Hyperledger Fabric network», *Stack Overflow*. [Em linha]. Disponível em: https://stackoverflow.com/questions/49875309/attempt-to-achieve-high-throughput-in-hyperledger-fabric-network. [Acedido: 06-Dez-2018].

[68] *BigchainDB Enhancement Proposals. Contribute to bigchaindb/BEPs development by creating an account on GitHub*. BigchainDB, 2018.

[69] «(PDF) Performance Evaluation of the Quorum Blockchain Platform». [Em linha]. Disponível em: https://www.researchgate.net/publication/327570196_Performance_Evaluation_of_the_Quorum_Blockchain_Platform. [Acedido: 06-Dez-2018].

[70] vasa, «Quorum stress-test 1: 140 TPS», *Hacker Noon*, 08-Mai-2018. [Em linha]. Disponível em: https://hackernoon.com/quorum-stress-test-1-140-tps-792f39d0b43f. [Acedido: 06-Dez-2018].

[71] K. Eroğlu, «Menapay Blockchain Tests: Quorum TPS», *MenaPay*, 03-Set-2018. .

[72] M. Ward, «Corda: Transactions Per Second (TPS)», *Corda*, 09-Fev-2018. .

[73] «When do you need blockchain? Decision models. – Sebastien Meunier – Medium». [Em linha]. Disponível em: https://medium.com/@sbmeunier/when-do-you-need-blockchain-decision-models-a5c40e7c9ba1. [Acedido: 05-Jan-2019].

[74] A. J. Steemann, «Bulk inserts in MongoDB, CouchDB, and ArangoDB», *ArangoDB*, 04-Set-2012. .

[75] «High Performance with Couchbase Server on Google Cloud», *The Couchbase Blog*, 28-Mai-2015. [Em linha]. Disponível em: https://blog.couchbase.com/couchbase-server-hits-1m-writes-with-3b-items-with-50-nodes-on-google-cloud/. [Acedido: 10-Jun-2019].

[76] D. Getz, «Datomic and the Myth of Slow Writes». [Em linha]. Disponível em: http://www.dustingetz.com/:datomic-myth-of-slow-writes. [Acedido: 11-Jun-2019].

[77] «Benchmarking Apache Kafka: 2 Million Writes Per Second (On Three Cheap Machines) | LinkedIn Engineering». [Em linha]. Disponível em: https://engineering.linkedin.com/kafka/benchmarking-apache-kafka-2-million-writes-second-three-cheap-machines. [Acedido: 05-Jan-2019].

[78] «The 5 fastest supercomputers in the world», *Science Node*. [Em linha]. Disponível em: https://sciencenode.org/feature/the-5-fastest-supercomputers-in-the-world.php. [Acedido: 08-Jan-2019].

[79] B. Schneier e D. Whiting, «A Performance Comparison of the Five AES Finalists». 07-Abr-2000.

[80] «Speed Comparison of Popular Crypto Algorithms». [Em linha]. Disponível em: https://www.cryptopp.com/benchmarks.html. [Acedido: 08-Jan-2019].

[81] hrasheed-msft, «High availability with Apache Kafka - Azure HDInsight». [Em linha]. Disponível em: https://docs.microsoft.com/en-us/azure/hdinsight/kafka/apache-kafka-high-availability. [Acedido: 14-Jan-2019].

[82] «Confluent REST Proxy API Reference — Confluent Platform». [Em linha]. Disponível em: https://docs.confluent.io/current/kafka-rest/api.html. [Acedido: 15-Jun-2019].

[83] «KSQL REST API Reference — Confluent Platform». [Em linha]. Disponível em: https://docs.confluent.io/current/ksql/docs/developer-guide/api.html. [Acedido: 15-Jun-2019].

[84] *Simple .NET logging with fully-structured events. Contribute to serilog/serilog development by creating an account on GitHub*. Serilog, 2019.

[85] «Amazon EC2 tipos de instâncias - AWS», *Amazon Web Services, Inc.* [Em linha]. Disponível em: https://aws.amazon.com/pt/ec2/instance-types/. [Acedido: 28-Jan-2019].

[86] «Likert Scale: What It Is & How to Use It», *SurveyMonkey*. [Em linha]. Disponível em: https://www.surveymonkey.com/mp/likert-scale/. [Acedido: 28-Jan-2019].

Anexo 1 *Access System* – Documentação da *API*

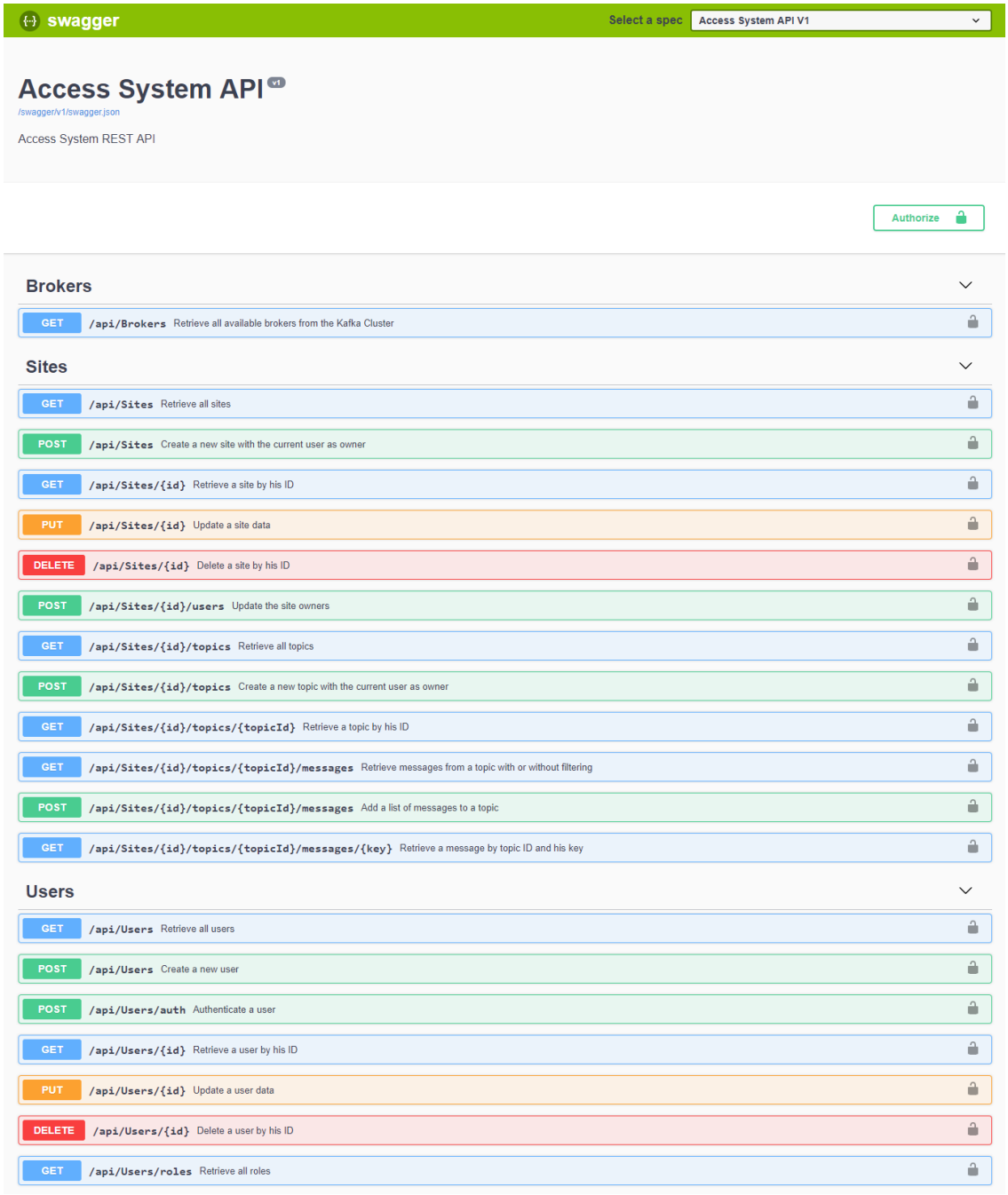


Figura 39: *Access System* – Documentação da *API* no *Swagger*

Anexo 2 *Access System* – Resultado dos testes unitários

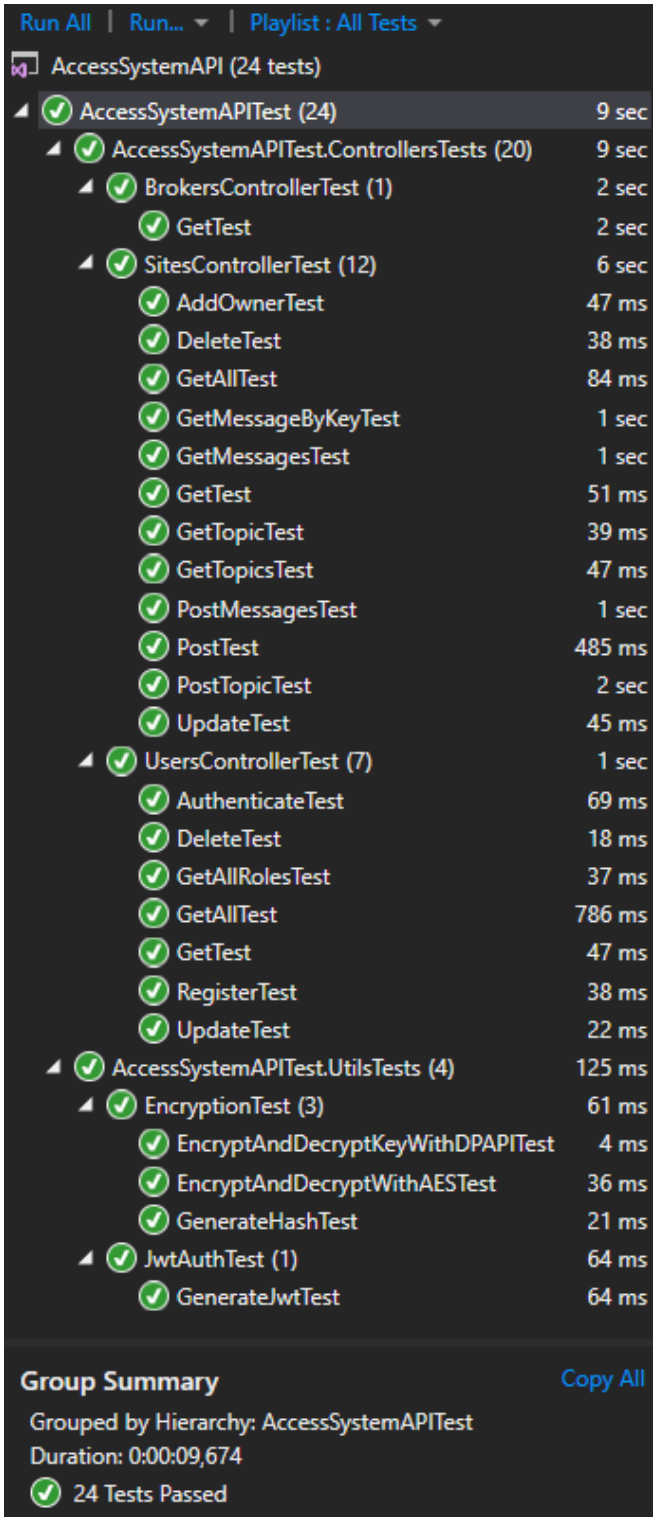


Figura 40: *Access System* – Resultado dos testes unitários

Anexo 3 *Web App – Mockups*

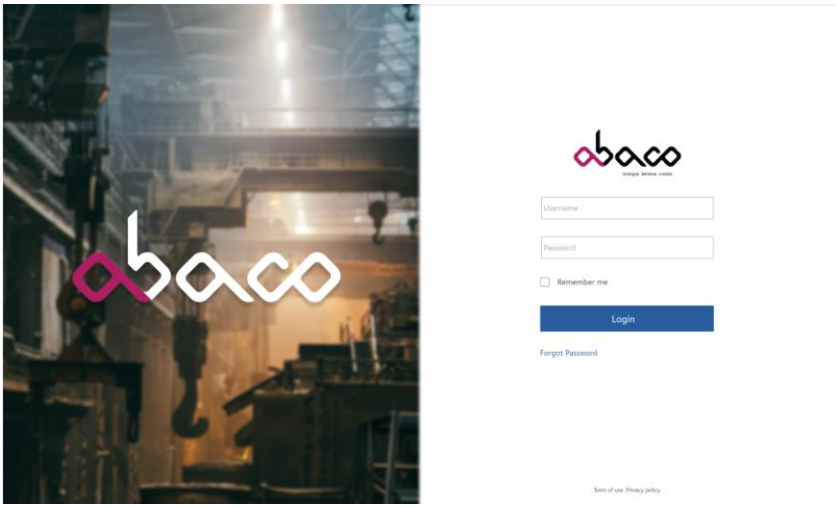


Figura 41: *Web App – Mockup* da página de *Login*

Figura 42: *Web App – Mockup* da página *Home*

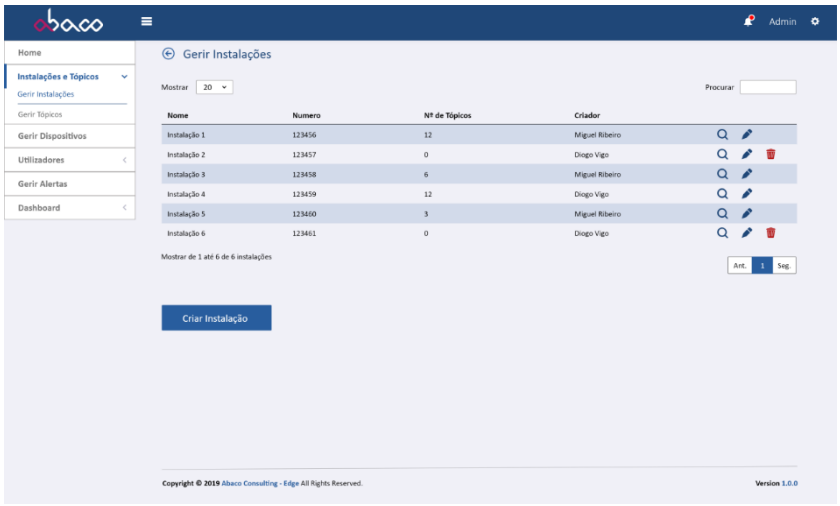


Figura 43: *Web App – Mockup* da página de gestão de instalações

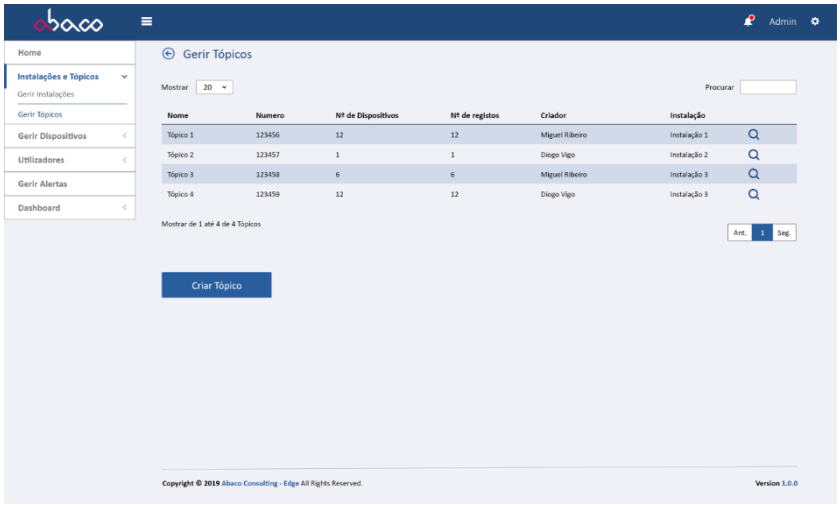


Figura 44: *Web App – Mockup* da página de gestão de tópicos

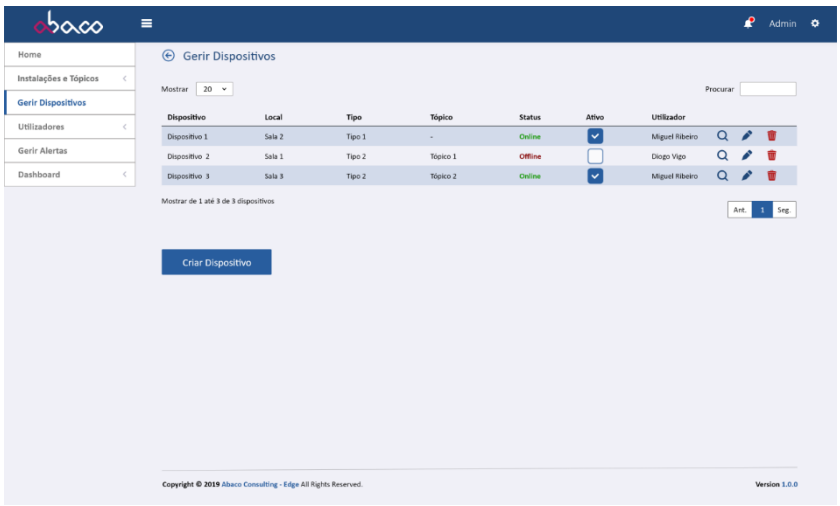


Figura 45: *Web App – Mockup* da página de gestão de dispositivos

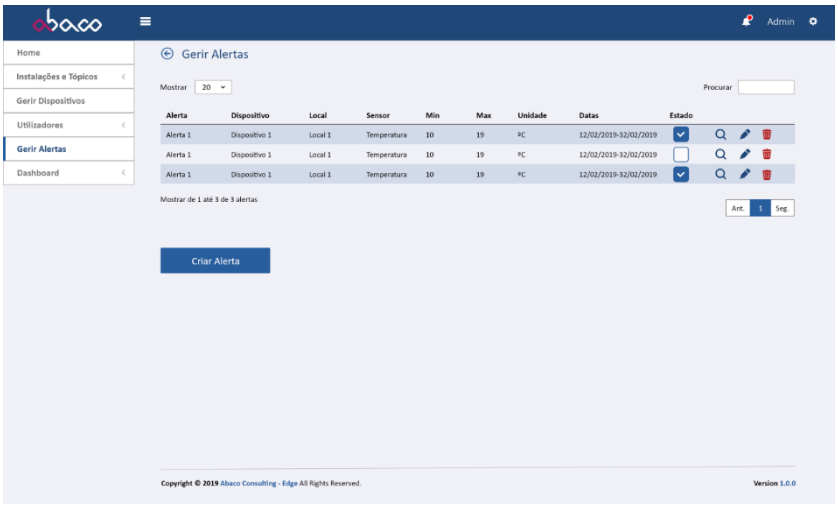


Figura 46: *Web App – Mockup* da página de gestão de alertas

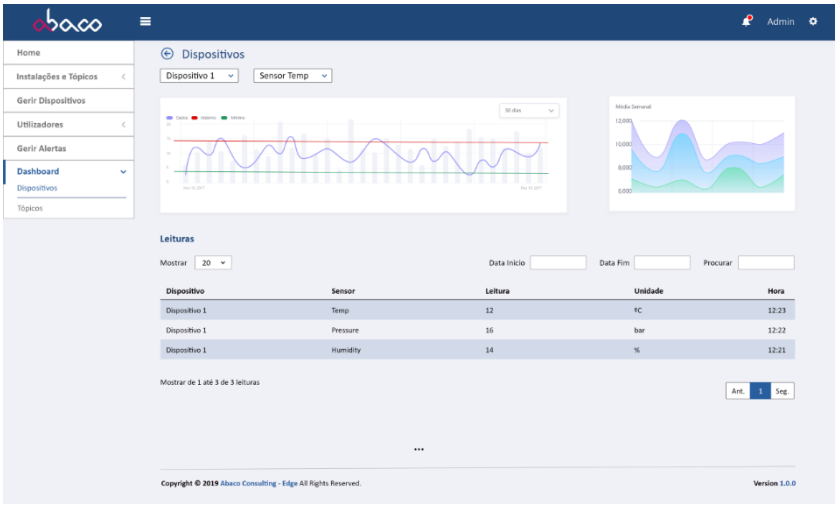


Figura 47: *Web App – Mockup* da página de *Dashboard* de dispositivos

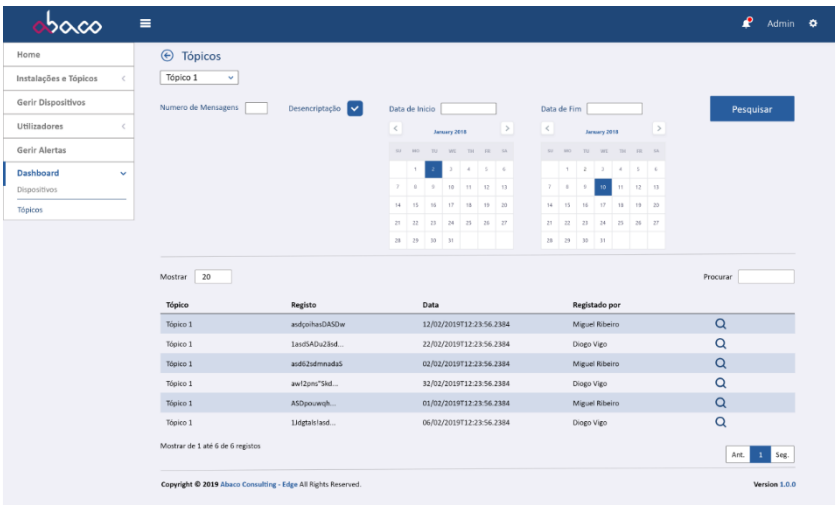


Figura 48: *Web App – Mockup* da página de *Dashboard* de tópicos

Anexo 4 *Web App* – Solução final

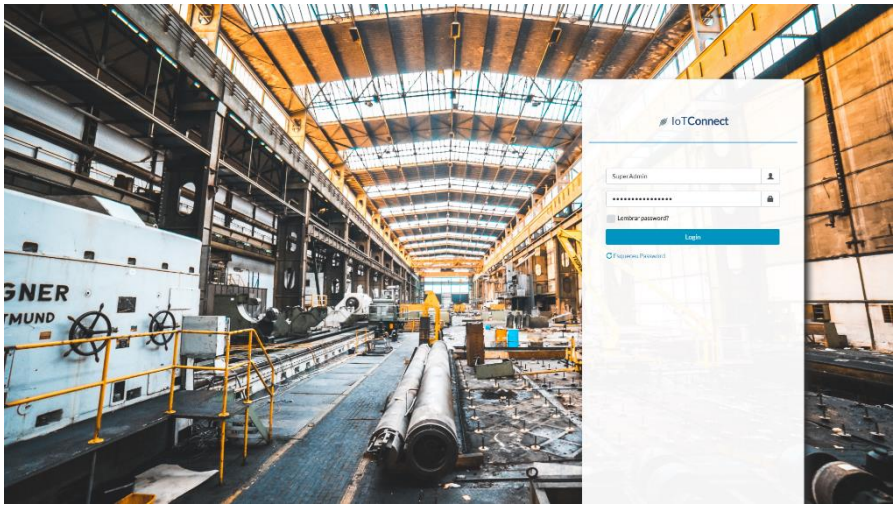


Figura 49: *Web App* – Página de *Login*

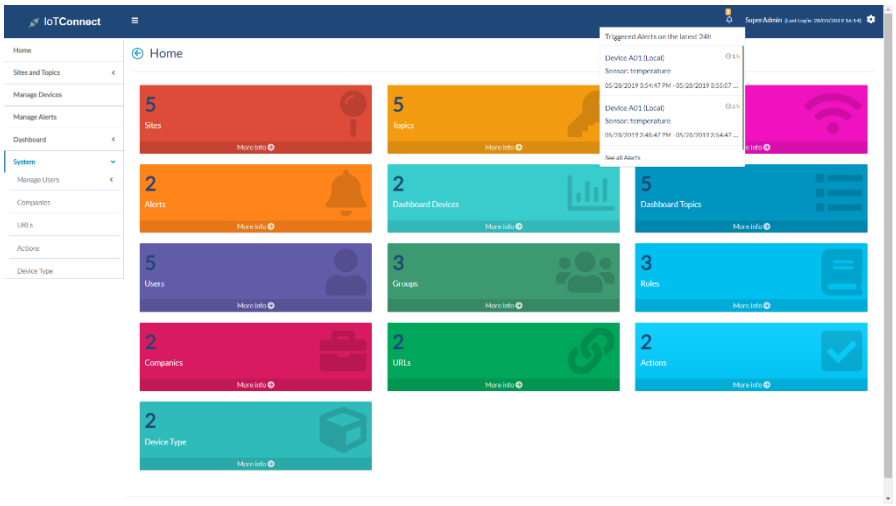


Figura 50: *Web App* – Página *Home*

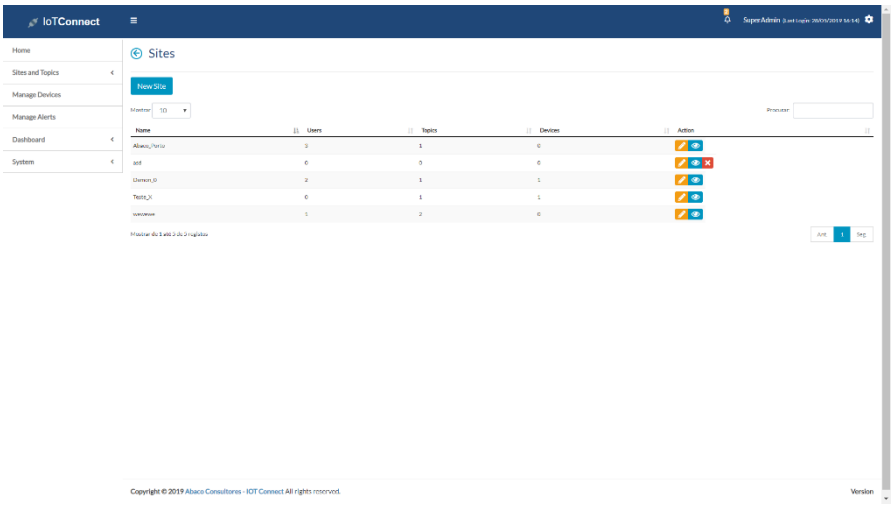


Figura 51: *Web App* – Página de gestão de instalações

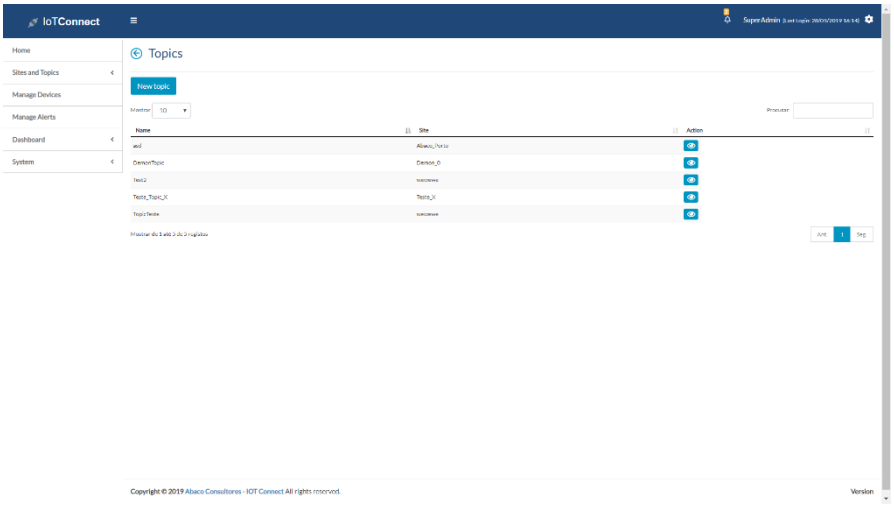


Figura 52: *Web App* – Página de gestão de tópicos

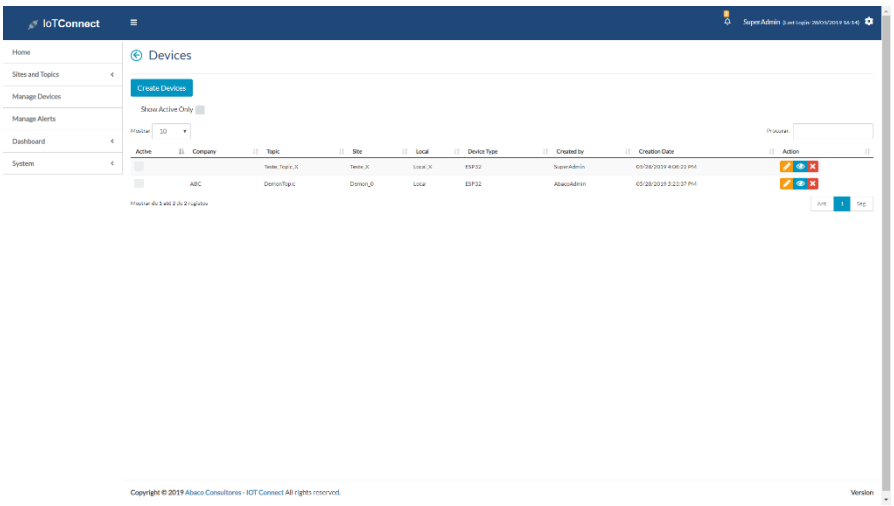


Figura 53: *Web App* – Página de gestão de dispositivos

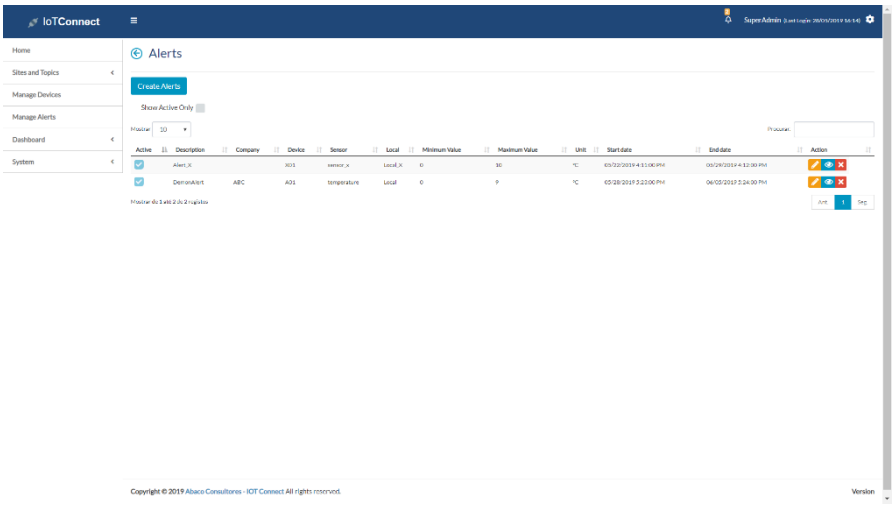


Figura 54: *Web App* – Página de gestão de alertas

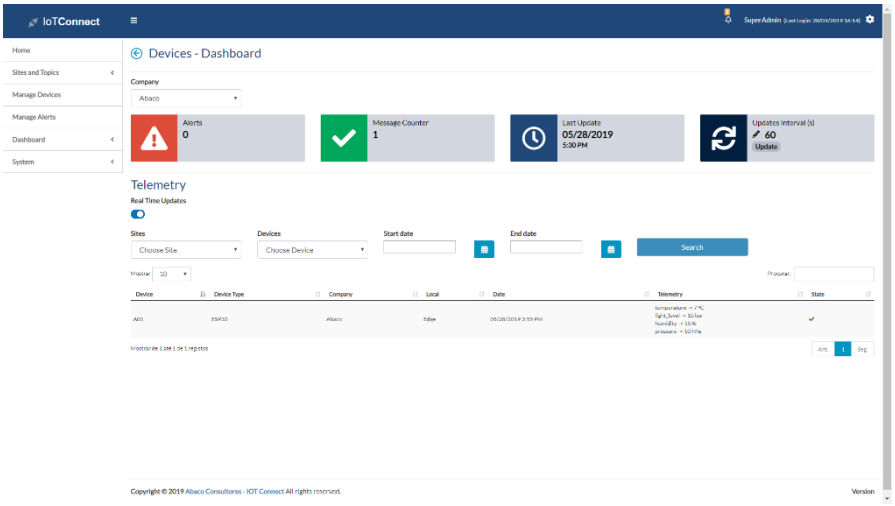


Figura 55: *Web App* – Página de *Dashboard* de dispositivos

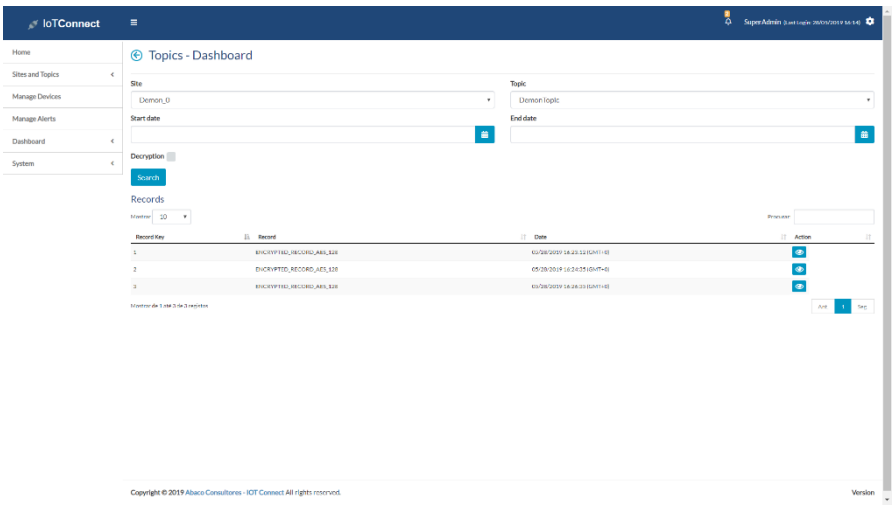
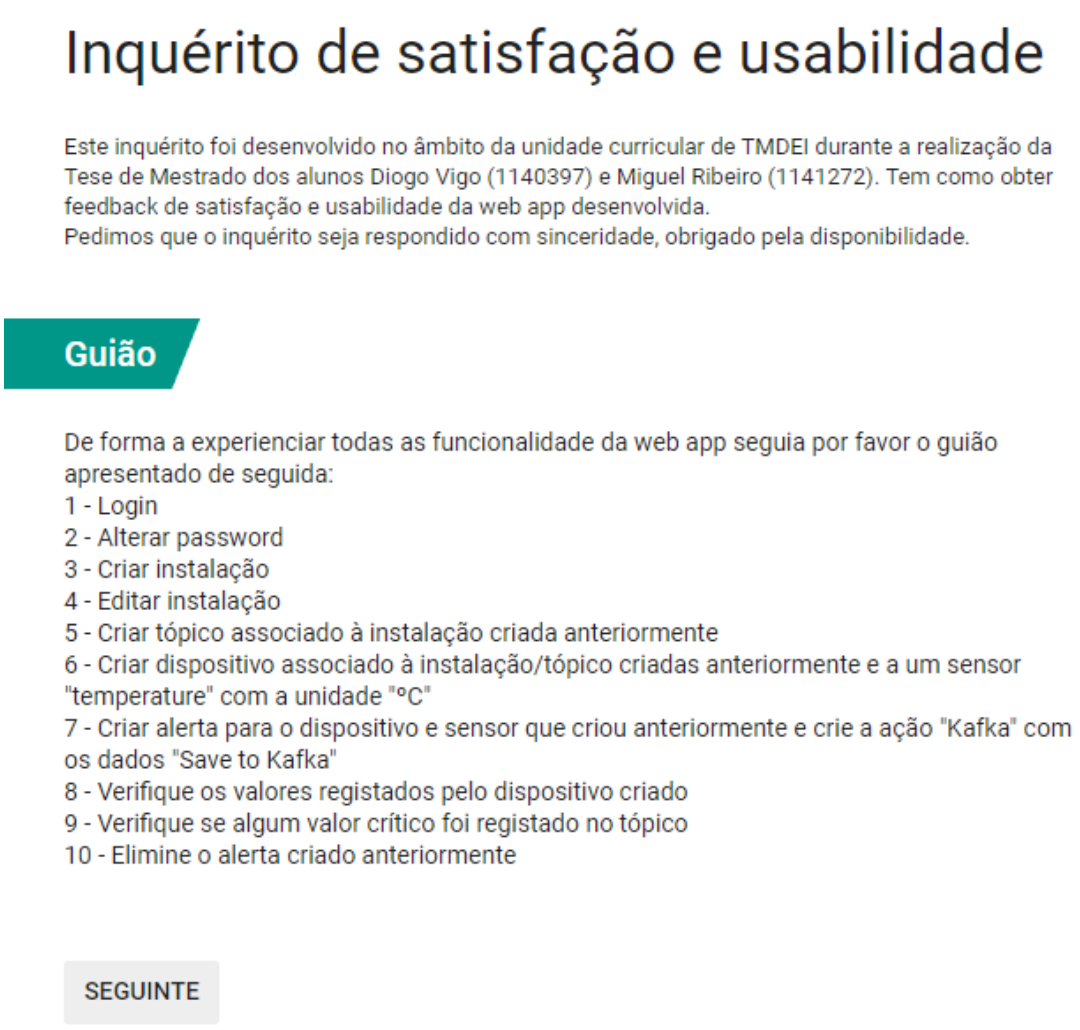


Figura 56: *Web App* – Página de *Dashboard* de tópicos

Anexo 5 *Web App* – Inquérito de satisfação e usabilidade



Inquérito de satisfação e usabilidade

Este inquérito foi desenvolvido no âmbito da unidade curricular de TMDEI durante a realização da Tese de Mestrado dos alunos Diogo Vigo (1140397) e Miguel Ribeiro (1141272). Tem como obter feedback de satisfação e usabilidade da web app desenvolvida.
Pedimos que o inquérito seja respondido com sinceridade, obrigado pela disponibilidade.

Guião

De forma a experienciar todas as funcionalidade da web app seguia por favor o guião apresentado de seguida:
1 - Login
2 - Alterar password
3 - Criar instalação
4 - Editar instalação
5 - Criar tópico associado à instalação criada anteriormente
6 - Criar dispositivo associado à instalação/tópico criadas anteriormente e a um sensor "temperature" com a unidade "ºC"
7 - Criar alerta para o dispositivo e sensor que criou anteriormente e crie a ação "Kafka" com os dados "Save to Kafka"
8 - Verifique os valores registados pelo dispositivo criado
9 - Verifique se algum valor crítico foi registado no tópico
10 - Elimine o alerta criado anteriormente

SEGUINTE

Figura 57: Inquérito de satisfação e usabilidade – Página nº 1

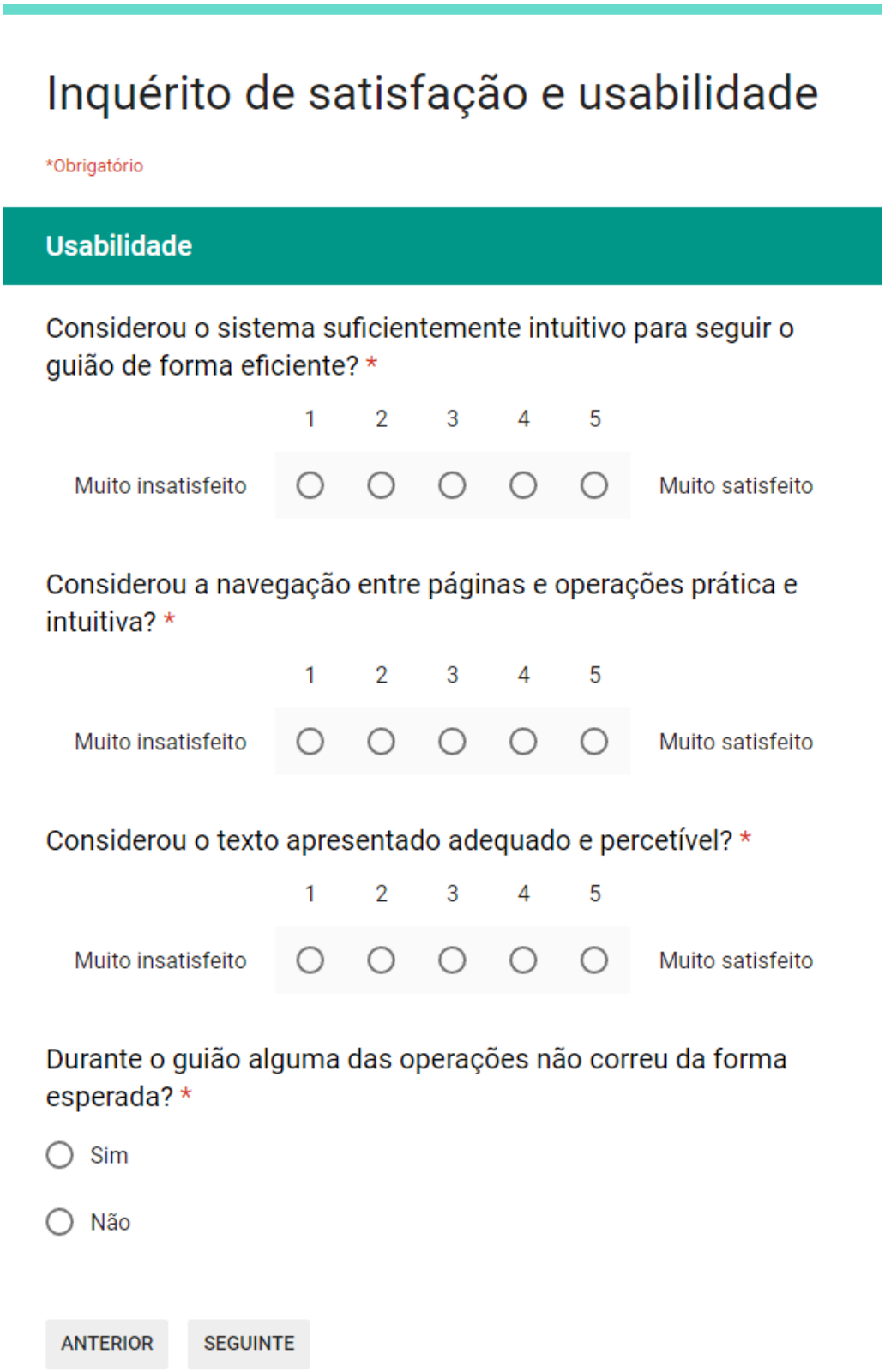


Figura 58: Inquérito de satisfação e usabilidade – Página nº 2

Inquérito de satisfação e usabilidade

*Obrigatório

Guião:
1 - Login
2 - Alterar password
3 - Criar instalação
4 - Editar instalação
5 - Criar tópico associado à instalação criada anteriormente
6 - Criar dispositivo associado à instalação/tópico criadas anteriormente e a um sensor "temperature" com a unidade "ºC"
7 - Criar alerta para o dispositivo e sensor que criou anteriormente e crie a ação "Kafka" com os dados "Save to Kafka"
8 - Verifique os valores registados pelo dispositivo criado
9 - Verifique se algum valor crítico foi registado no tópico
10 - Elimine o alerta criado anteriormente

Especifique a(s) operação(s) em que tal situação ocorreu. *

- [ ] 1
- [ ] 2
- [ ] 3
- [ ] 4
- [ ] 5
- [ ] 6
- [ ] 7
- [ ] 8
- [ ] 9
- [ ] 10

ANTERIOR SEGUINTE

Figura 59: Inquérito de satisfação e usabilidade – Página nº 3

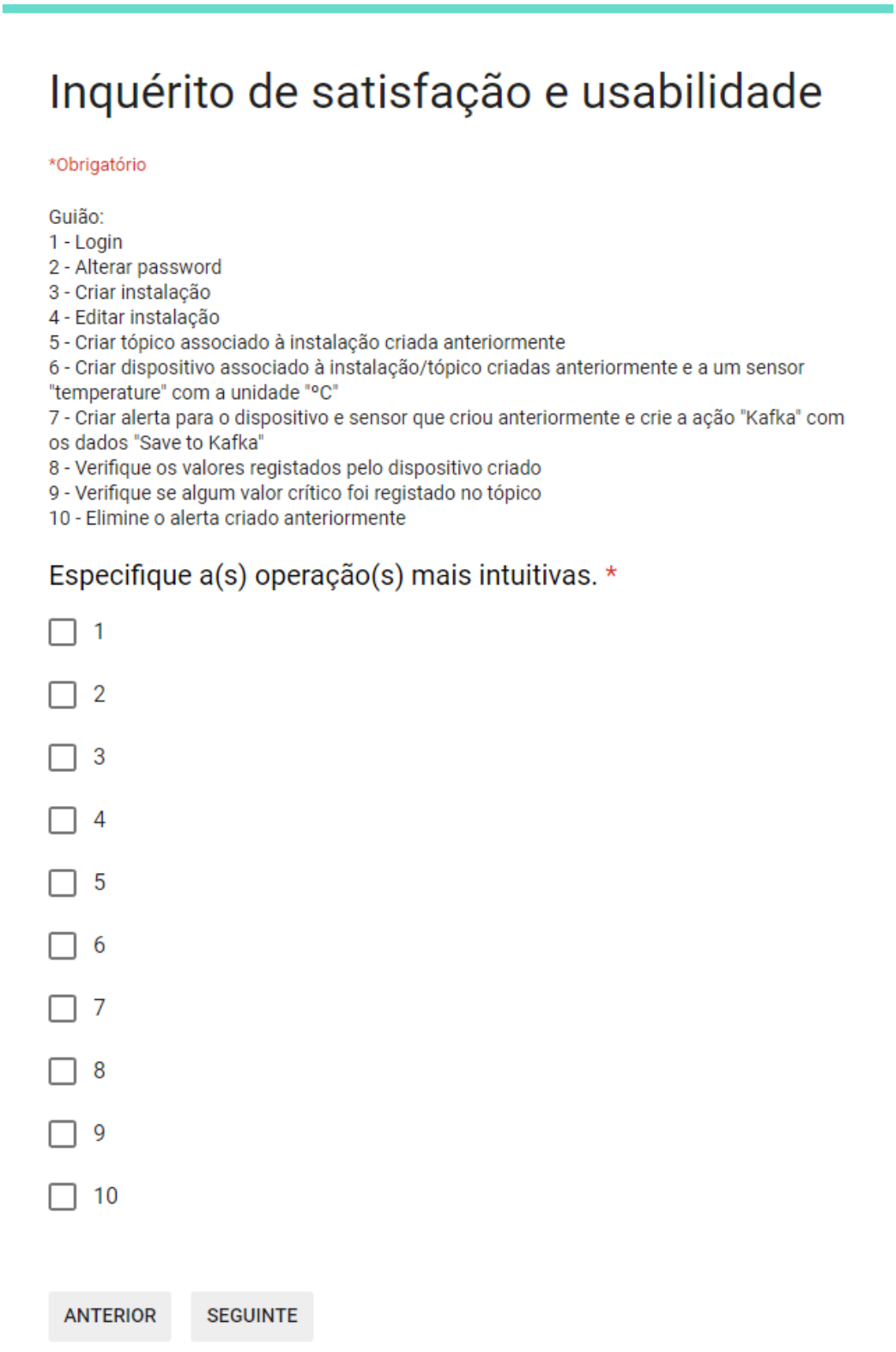


Figura 60: Inquérito de satisfação e usabilidade – Página nº 4

Inquérito de satisfação e usabilidade

*Obrigatório

Guião:
1 - Login
2 - Alterar password
3 - Criar instalação
4 - Editar instalação
5 - Criar tópico associado à instalação criada anteriormente
6 - Criar dispositivo associado à instalação/tópico criadas anteriormente e a um sensor "temperature" com a unidade "ºC"
7 - Criar alerta para o dispositivo e sensor que criou anteriormente e crie a ação "Kafka" com os dados "Save to Kafka"
8 - Verifique os valores registados pelo dispositivo criado
9 - Verifique se algum valor crítico foi registado no tópico
10 - Elimine o alerta criado anteriormente

Especifique a(s) operação(s) menos intuitivas. *

- [ ] 1
- [ ] 2
- [ ] 3
- [ ] 4
- [ ] 5
- [ ] 6
- [ ] 7
- [ ] 8
- [ ] 9
- [ ] 10

ANTERIOR SEGUINTE

Figura 61: Inquérito de satisfação e usabilidade – Página nº 5

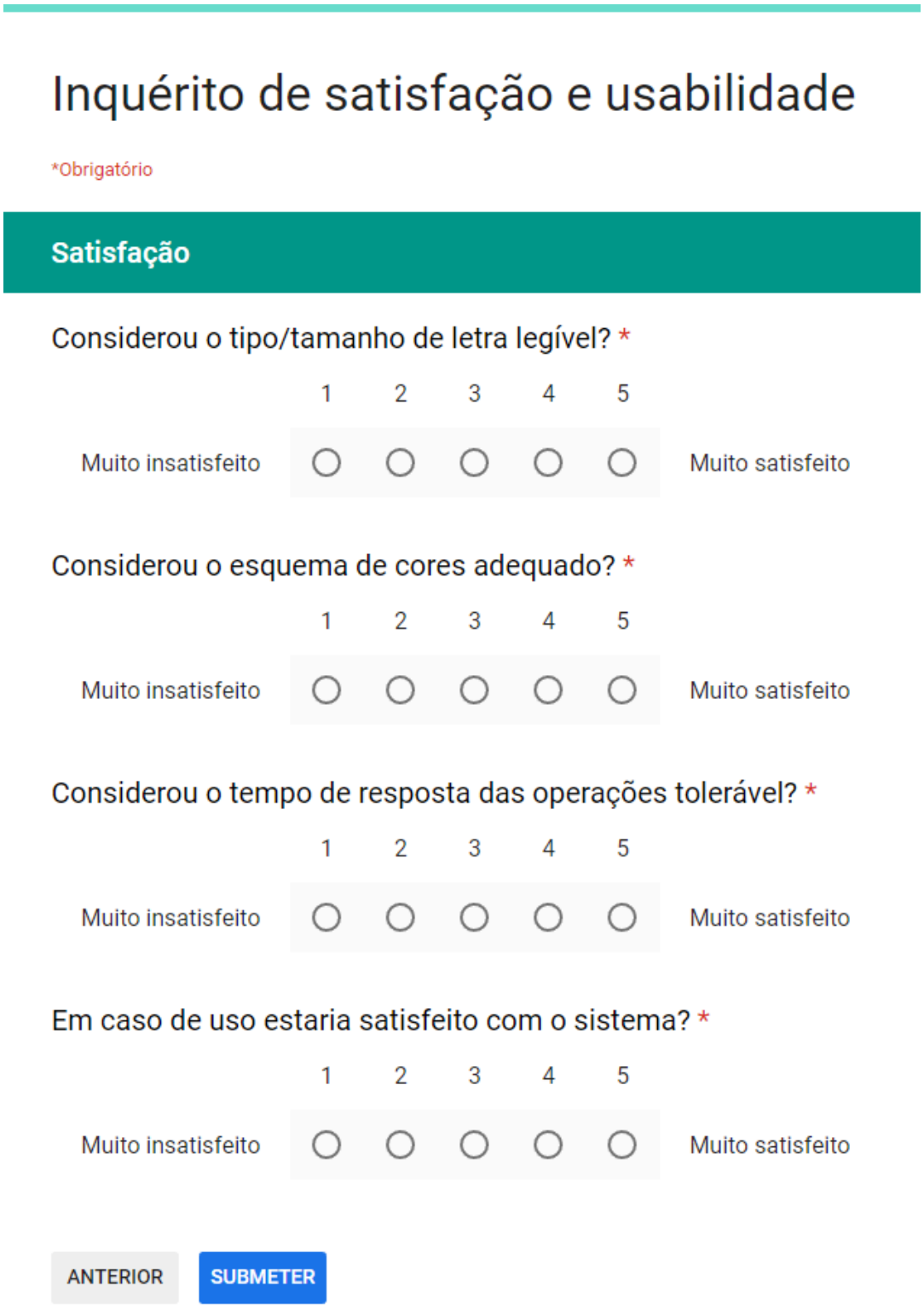


Figura 62: Inquérito de satisfação e usabilidade – Página nº 6